博士論文

コンピュータグラフィックスにおける流体映像の効率的な生成に関する研究

北海道大学 大学院情報科学研究科
メディアネットワーク専攻

佐藤　周平

目 次

図 目 次

表 目 次

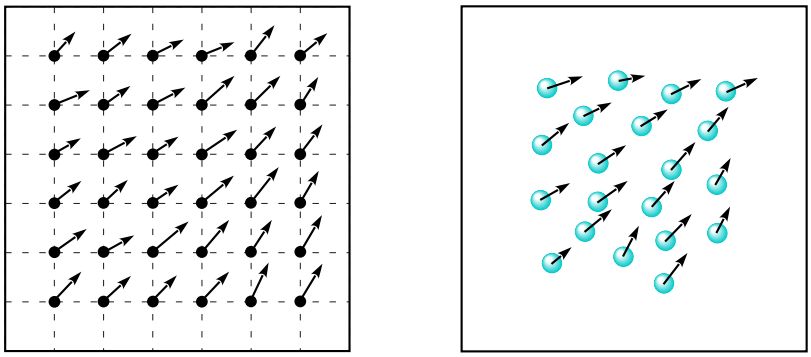
第1章　序論

本章では，コンピュータグラフィックス (CG) において，これまでに研究されている流体解析手法に関して概説を行う．また，本研究の位置づけを明確にした上で，流体映像を効率的に生成するために本論文において提案する 3 つの手法に関して，関連する従来手法の議論とともにその概要について説明する．

1.1　研究背景

近年，コンピュータの普及，性能の向上などによって CG の技術が様々な分野に応用されている．その応用分野の中でも第一に挙げられるのが映画やコンピュータゲームなどにおける映像生成である．また，その他にも建築物・工業デザインなどの支援システムである CAD（Computer Aided Design）や気象・自然災害などのシミュレーションシステム，医療分野における臓器・骨格の可視化システムなど，多岐に渡って CG 技術が利用されている．このような CG 技術の用途拡大に伴い，CG の分野では多種多様な研究が行われている．中でも自然現象のビジュアルシミュレーションに関する研究は重要な研究課題のひとつとして位置づけられており，映像表現におけるリアリティの向上や演出の多様性に大きく貢献している．

自然現象のビジュアルシミュレーションに関する研究では，煙，水，炎といった流体現象に注目した研究が盛んに行われており，現在までにそれらの自然現象を表現する手法が数多く提案されている．その中でも，自然現象の形状や動きなどを写実的に表現する場合，それらの物理現象を解析する方法が特に有効である．従来の研究では数値流体解析を利用することで物理現象を忠実にシミュレーションし，極めてリアルな映像を生成することができる [9][12][25].



(a) 格子法　　(b) 粒子法

図 1.1: 数値流体解析手法

数値流体解析手法には大きく分けて格子法と粒子法が存在する．格子法では，図 1.1(a) に示すようにシミュレーション空間を格子状に分割することで流体の動きを解析する．具体的には，格子点と任意のタイムステップを設定し，流れ場を空間および時間において一定間隔ごとにサンプリングすることで，流体の支配方程式を空間的，時間的に離散化し数値解析を行う．代表的な解析法には差分法や有限要素法などがある．一方，粒子法では，流体を粒子の集まりとして近似することで流体の動きを解析する．具体的には，図 1.1(b) に示すように，シミュレーション空間に多数の速度・座標を持った粒子を発生させ，粒子間の相互作用を考慮することで流体の支配方程式の数値解析を行う．代表的な解析手法には，SPH（Smoothed Particle Hydrodynamics）法や MPS（Moving Particle Semi-implicit）法などが存在する．本論文では，離散化の容易性，計算の高速性，また多くの流体現象に課される非圧縮性の法則の適用の容易性から，格子法を用いた流体シミュレーションを対象とする．そして，格子法による流体シミュレーションを用いて写実的な流体映像を効率的に生成するための手法を提案する．

CG において実用的な格子法による流体シミュレーション手法は，Stam によって 1999 年に提案された [35]．セミ・ラグランジュ法を用いることで，Navier-Stokes 方程式に存在する非線形項（移流項）を安定的に解くことができる．Stam の手法以降，

様々な流体現象を対象とした数多くの解析手法が提案されている．それらの手法の詳細は [3] にまとめられている．前述したように，これら数値流体解析による手法は写実的な映像生成には大変有効である．しかし，シミュレーションには様々な物理パラメータが依存している．そのため，映画やゲームなどのエンターテインメントアプリケーションにおいて，ユーザが意図した動きや形状となるような流体映像をシミュレーションにより得るためには，試行錯誤的に物理パラメータを調節しなければならず，これは大変煩雑な作業である．また，流体シミュレーションは計算コストが高く，ユーザが所望かつ高品質な映像を作るためには，多大な時間と労力が必要となってしまう．

次節では，上記問題を解決するために，本論文で提案する 3 つの手法について研究目的と手法の概要について述べる．また，提案手法に関連する従来のアプローチと，それらのアプローチに対する提案手法の位置づけについても述べる．

1.2 研究概要

前節でも述べたとおり，流体シミュレーションを用いることにより，CG において写実的な映像を作成することができる．近年では，映画やゲームにおいて，流体シミュレーションを用いて作成された水，炎などの写実的な映像が数多く登場し，シーンのリアリティ向上に貢献している．しかし，実写レベルの流体映像をシミュレーションにより作成する場合，非常に高い計算コストがかかってしまう．前節で述べたように，流体シミュレーションでは，格子または粒子の集合として流体を表現することで，その動きの解析を行う．実写レベルの映像の場合，流体の動きを高精度に捉えるために，非常に多くの数の格子や粒子が必要になってしまう．これにより，1 つの映像の作成にも膨大な時間が必要となってしまう．さらに，映画やゲームでは，作成するシーンに合った流体映像や 1 つのシーンに多数の流体映像を必要とする場合がある．このような場合，映像を制作するアニメータは，流体シミュレーションに存在するパラメータを何度も変更しながら，シミュレーションを繰り返さなければならない．しかし，リアルな映像の生成には，上記でも述べたように，多くの計算時間を必要とする．そのた

め，パラメータの変更とシミュレーションを何度も繰り返すことは，非常に時間のかかる煩雑な作業であり，映像制作期間の長期化を招く原因となってしまう．このようなことから，流体シミュレーションによる計算，パラメータ調整などの作業といった流体映像生成に現存する問題を解決するための研究が世界的にも注目されている．本論文では，上記問題を解決し，ユーザが所望の流体映像を効率的に生成することを可能とするため，以下の 3 つの手法を提案する．

- 爆発シミュレーションの制御手法
- 流体シミュレーションの超解像処理
- 流れ場のバリエーション生成

以下では，上記問題点の詳細と関連する従来研究に対する各手法の位置付けについて説明する．

1.2.1 爆発シミュレーションの制御

映画やゲームなどの映像作品において，爆発現象の映像はよく用いられ，特に戦争映画や SF 映画などの戦闘シーンでは，爆発映像が数多く描写される．これら映像制作において爆発現象を用いる際，実際の爆発を利用することは危険を伴うため，爆発のシミュレーションにより CG として製作される場合が多い．そのため，流体シミュレーションを用いて爆発現象を生成するための手法が提案されている [10, 14, 17, 32]．これらの手法を用いることで，映画やゲームにおいて爆発やそれに類似した現象の写実的なアニメーションを生成できる．しかし前述したように，ユーザが所望のアニメーションを作成する場合，シミュレーションに存在する数多くのパラメータを試行錯誤的に調整しなければならない．また，物理パラメータの調整のみで，完全にユーザの意図した結果を得るのは極めて困難である．

一方，映画やゲームなどのアプリケーションでは，流体を特定の形状として表現したいという要求も存在する．しかし，既存の流体シミュレーションのパラメータ調整

のみで，流体をある特定の形状として表現することは非常に困難である．この問題を解決するために，シミュレーションを制御することで，所望の形状として流体を表現するための手法がいくつか提案されている [6, 8, 15, 18, 20, 24, 34, 36, 37]．これらの手法には，主に 2 つのアプローチが存在する．1 つは，シミュレーションに対し現実には存在しないような外力を作用させることで流体の動きを制御するという方法である．このアプローチにより，煙や水を文字や動物など現実ではありえないような形状として表現することが可能である．もう一方は，シミュレーションに存在するパラメータを自動で調整するというアプローチである．代表的なものには，流体の目標となる形状と現在形状との差分を物理パラメータにフィードバックするといった方法が提案されている．このようなアプローチにより，現実にはない外力を加えた場合に比べ，リアルな動きを保ったまま，形状を制御することができる．しかし，これらの制御手法は，主に煙や水を対象にしたものであり，炎など燃焼を伴った流体を制御するための一般的な手法は現状存在しない．燃焼を伴う流体では，燃焼過程におけるランダム要素や熱による浮力のため，煙などの他の流体と比較し動きや形状を制御することが難しい．一方，手続き的に所望の炎アニメーションを生成するための方法がいくつか提案されている [13, 21]．これらの手法では，炎の通る経路を表わす曲線を任意に編集することで，任意の炎アニメーションを作成することができる．しかし，これらの方法では，流体シミュレーションを用いていないため，物理法則に沿ったシミュレーションにより作成された結果と比べ，写実性に乏しい結果となってしまう．これまで述べたように，煙，水，雲および炎に関しては，所望の映像を作成するための方法が提案されている．しかし，爆発を対象とした所望の映像を得るための一般的な方法はいまだに存在しない．

そこで本研究では，燃焼を伴った流体現象として，爆発現象に着目し，爆発を所望の形状として表現するための制御手法を提案する．爆発現象は，発生時に急速な熱膨張により大きな速度が発生し，また熱による浮力も発生するため，パラメータを調整して動きを制御することは非常に難しい．また，非常に大きな速度が発生するため，従

来手法をそのまま適用した場合，動きが不自然になってしまう．そのため，提案法では，最適化と予測制御を用いることで，爆発の制御を可能とする．提案法により，これまで困難であった爆発の制御を実現し，特定の形状として表現された爆発の映像を作成する際のパラメータ調整にかかる試行回数を大幅に削減する．

1.2.2 流体シミュレーションの超解像処理

上述の制御手法を用いることで，所望の形状として流体を表現することが可能となり，物理パラメータの調整にかかる試行回数の削減もできた．しかし，制御手法を用いた場合でも，いくつかの物理パラメータや制御パラメータの調整が必要である．前述したように，流体シミュレーションは計算コストが高く，実写レベルの映像を作成するための精度の高いシミュレーション（高解像度のシミュレーション）において何度も調整を行うことは，大変労力と時間のかかる作業である．この問題を解決するために，精度は低いが高速に計算可能なシミュレーション（低解像度のシミュレーション）において所望の流体の動きや形状を作成し，その動きを保ったまま高解像度の結果へ変換するというアプローチに注目が集まっている．この考え方を基に，高解像度のシミュレーションを低解像度のシミュレーション結果に従うよう制御する手法がいくつか提案されている [26, 41]．しかし，これらの手法では最終的な結果のクオリティを確認するために，高解像度のシミュレーションを実行しなければならなず，実写レベルの映像を作る場合計算コストが高い．一方，ノイズ関数と流体シミュレーションを組み合わせた手法も提案されており [19, 29, 33]，高解像度のシミュレーションを実行せずに詳細な動きを含んだアニメーションを作成することができる．しかし，ノイズにより詳細を付加しているため，これらの手法により生成される結果は，物理シミュレーションで得られる結果よりもノイズ感の強い結果となってしまう．

本研究においても高解像度のシミュレーションを実行することなく，低解像度のアニメーションから高品質な結果を作成する．ただし本手法では，前計算により得られた流体シミュレーション結果の速度場を利用して，高解像度のアニメーションを超解

像的に生成する．物理シミュレーションによる結果を利用するため，ノイズ関数を用いた手法と比較して写実性を向上させることができる．また，前計算で用意する速度場は，2 次元のシミュレーションにより生成するため，計算コストも削減することができる．本提案手法により，所望のリアルな流体映像を効率的に作成することができる．

1.2.3　流れ場のバリエーション生成

これまでの 2 つの提案手法では，一つの流体映像を作成する場合に，所望の映像を効率的に生成するための手法を提案した．一方，映画やゲームなどの映像制作において，1 つのシーンに複数の炎や爆発が存在するような場合がある．例えば，村全体が火事になっているようなシーンや戦争映画などでミサイルにより複数の爆発が起こっているシーンなどが例としてあげられる．このような複数の流体映像を含む大規模なシーンでは，同一のアニメーションを繰り返し用いると映像のリアリティが低下してしまう．そのため，類似しているが動きの異なる複数の流体アニメーションを用いる必要がある．これは，異なるパラメータ設定で何度も流体シミュレーションを繰り返すことで作成が可能である．しかし，シミュレーションパラメータを調整して，そのような類似したアニメーションを作成することは，非常に困難である．また，前述したように流体シミュレーションのコストが高いため，前節までのアプローチを用いたとしても，全てのアニメーションを作るまでには膨大な計算コストと時間が必要である．このような場合，ノイズ関数を利用した手続的な手法 [4, 13, 21, 28] を用いることで，比較的低コストで動きの異なる複数のアニメーションを作成することができる．しかし，これらの手法では，流体シミュレーションを用いていないため，物理法則に沿ったシミュレーションを用いた場合よりも写実性に乏しい映像となってしまう．また，一つ一つの映像を手作業で作成するのは，非常に手間のかかる作業である．

そこで本研究では，流体シミュレーションにより生成された単一のシミュレーションデータから，シミュレーションを実行せずに，流れ場の様々なバリエーションを生成する手法を提案する．これは，流体の流れ場を非圧縮性を持った基底関数で展開す

ることで実現する．また提案法では，周波数および空間領域でのバリエーションを生成する．提案法により，大規模なシーンにおいて，複数の流体映像を効率的に生成することが可能である．

1.3 論文の構成

本節では，本論文の構成について説明する．本章は，CG 分野における流体映像生成に関する研究課題や近年における CG 技術の応用範囲について述べ，本研究に関する研究背景およびその背景に基づいて本論文において提案する手法の概要について説明した．第 2 章では，本論文の提案手法が取り扱う格子法による流体シミュレーションの方法について解説する．次に，第 3 章では，爆発シミュレーションの制御手法，第 4 章では流体シミュレーションの超解像処理，第 5 章では流体シミュレーションのバリエーション生成手法について，研究背景および関連研究について詳しく説明する．また，それぞれの提案手法の詳細と実験結果，それに対する考察について述べる．最後に，第 6 章で，本論文の結びとする．

第2章　格子法による流体シミュレーション

本章では，本論文が取り扱う，格子法による流体シミュレーションの方法について説明する．

まず，シミュレーション空間を格子状に分割し，各格子点に速度 $\mathbf{u}$ および密度 D，炎や爆発の場合は加えて温度 T も割り付ける．そして，それらの値の時間変化を非圧縮性 Navier-Stokes 方程式に従って計算する．本章の各節において，速度および密度，温度の解析方法，本論文で考慮する外力について解説する．

2.1　速度場の解析方法

速度場の解析は，以下の方程式に従って計算される．

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u}\cdot\nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \nu\nabla^2\mathbf{u} + \mathbf{f} \tag{2.1}$$

$$\nabla\cdot\mathbf{u} = 0 \tag{2.2}$$

p は圧力，ν は動粘性係数，$\mathbf{f}$ は外力である．式 (2.1) および (2.2) は，非圧縮性条件下における Navier-Stokes 方程式である．非圧縮性流体を扱う場合のみ式 (2.2) が成り立ち，この式は連続の式と呼ばれる．また，式 (2.1) は速度場の時間発展を表す方程式であり，右辺第 1 項から，移流項，圧力項，拡散項，外力項と呼ばれる．以下では，この微分方程式の計算方法について説明する．簡単のため，2 次元のシミュレーション空間を対象とする．

まず，式 (2.1) を時間に関して離散化し，以下の式を得る．

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \Delta t(-(\mathbf{u}_n\cdot\nabla)\mathbf{u}_n - \frac{1}{\rho}\nabla p + \nu\nabla^2\mathbf{u}_n + \mathbf{f}) \tag{2.3}$$

ここで，$\mathbf{u}_n$ は n ステップ目の速度場，$\mathbf{u}_{n+1}$ は $\mathbf{u}_n$ に対し式 (2.1) を解いて得られる $n+1$ ステップ目の速度場である．また，Δt は 1 ステップあたりの時間幅を表す．この式 (2.3) は，非線形項（移流項）を含む等，複雑な方程式であるため，直接解を求めることはできない．そのため，外力項，拡散項，移流項，圧力項の順序で，項ごとに差分法を用いて近似的に計算を行う．

まず，外力項は以下のように差分法により計算し，速度場 $\mathbf{u}_n^1$ を得る．

$$\mathbf{u}_n^1 = \mathbf{u}_n + \Delta t \mathbf{f} \tag{2.4}$$

次に，拡散項を計算し $\mathbf{u}_n^2$ を得る．

$$\mathbf{u}_n^2 = \mathbf{u}_n^1 + \Delta t \nu \nabla^2 \mathbf{u}_n^1 \tag{2.5}$$

上式は，ある格子点 (i, j) に注目した場合，以下のように離散化される．

$$\begin{aligned}\mathbf{u}_n^2(i,j) &= \mathbf{u}_n^1(i,j) \\ &+ \nu \frac{\Delta t}{h^2}(\mathbf{u}_n^1(i+1,j) + \mathbf{u}_n^1(i-1,j) + \mathbf{u}_n^1(i,j+1) + \mathbf{u}_n^1(i,j-1) - 4\mathbf{u}_n^1(i,j))\end{aligned} \tag{2.6}$$

これに対し陰解法を適用し，最終的に以下のような方程式とする．

$$\begin{aligned}\mathbf{u}_n^1(i,j) = (1 + 4\nu \frac{\Delta t}{h^2})\mathbf{u}_n^2(i,j) \\ - \nu \frac{\Delta t}{h^2}(\mathbf{u}_n^2(i+1,j) + \mathbf{u}_n^2(i-1,j) + \mathbf{u}_n^2(i,j+1) + \mathbf{u}_n^2(i,j-1))\end{aligned} \tag{2.7}$$

この方程式が格子点の数だけ立つので，式 (2.5) は $\mathbf{Ax} = \mathbf{b}$ の連立一次方程式の形となる．そして，この連立一次方程式を，ガウス・ザイデル法を用いて解くことで，拡散項を計算した場合の解を算出する．

続いて移流項の計算は，以下のような式となる．

$$\mathbf{u}_n^3 = \mathbf{u}_n^2 + \Delta t(-(\mathbf{u}_n^2 \cdot \nabla)\mathbf{u}_n^2) \tag{2.8}$$

この式を差分法により計算した場合，Δt をかなり小さくとらなければ計算が安定しない．しかし，CG 分野における流体解析では，厳密な計算精度よりも計算速度が重視

されるため，差分法が安定するほど小さい値は必要とされない．これに対し，ある程度 Δt を大きくとった場合でも，移流項を安定的に計算可能なセミ・ラグランジュ法が提案された [35]．本論文でも移流項の計算には，セミ・ラグランジュ法を用いる．

最後に，圧力項を以下の式により計算する．

$$\mathbf{u}_{n+1} = \mathbf{u}_n^3 - \Delta t \frac{1}{\rho} \nabla p \tag{2.9}$$

ここで，圧力の値が未知であるため，圧力を算出するための方程式を導く必要がある．そこでまず，式 (2.9) の両辺に ∇ を作用させ，結果の速度場 $\mathbf{u}_{n+1}$ が式 (2.2) を満たすという条件を課すことで，圧力に関する以下の方程式を導出する．

$$\frac{\Delta t}{\rho} \nabla^2 p = \nabla \mathbf{u}_n^3 \tag{2.10}$$

この式をある格子点 (i, j) において，以下のように空間に関して離散化する．

$$\frac{\Delta t}{\rho h^2}(p(i+1,j) + p(i-1,j) + p(i,j+1) + p(i,j-1) - 4p(i,j)) = \frac{1}{2h}(u_n^3(i+1,j) - u_n^3(i-1,j) + v_n^3(i,j+1) - v_n^3(i,j-1)) \tag{2.11}$$

ここで，u，v はそれぞれ 2 次元空間における速度 $\mathbf{u}$ の水平方向成分と鉛直方向成分を表わす．この式も式 (2.5) と同様格子点の数だけ方程式が立ち，連立一次方程式の形となる．そのため，ガウス・ザイデル法により解を計算する．これにより得られた圧力 p を式 (2.9) に適用し，速度場 $\mathbf{u}_{n+1}$ を得る．

2.2 スカラー場の解析方法

密度場および温度場の解析方法について説明する．まず，密度場の時間発展は式 (2.1) により計算された速度場を用いて次式により表わされる．

$$\frac{\partial D}{\partial t} = -(\mathbf{u} \cdot \nabla) D + \nu \nabla^2 D + D_s \tag{2.12}$$

ここで右辺第 1 項は密度の移流を，第 2 項は密度の拡散を表している．また，D_s は，流体の発生源から供給される密度量を表わしている．この密度場の解析については，

速度場と同様差分法により時間に関して離散化し，計算する．各項の計算については，右辺第 1 項からそれぞれ速度場における移流項，拡散項，外力項と同様の方法で数値的に計算する．

次に，温度場の解析は次式を計算することで行われる．

$$\frac{\partial T}{\partial t} = -(\mathbf{u} \cdot \nabla)T - c_r(\frac{T - T_{amb}}{T_{max} - T_{amb}})^4 + c_k \nabla^2 T + T_s \tag{2.13}$$

ここで c_r は冷却定数，T_{amb} は環境温度である．また，T_{max} はシミュレーション空間内での最大温度であり，シミュレーションにより得られた値を設定する．右辺第 1 項は温度の速度場による移流を，右辺第 2 項は流体温度の環境への放射損失を表している．また，c_k は熱伝導率を表しており，右辺第 3 項は温度の拡散項となる．最後に T_s は，熱源から供給される温度量を表わしている．温度場の数値解析も，速度場と同様差分法により時間に関して離散化し，各項ごとに計算を行う．また，第 1 項は速度場の移流項，第 3 項は拡散項と同様の方法で計算する．第 2 項および第 4 項は，速度場の外力項と同様の方法で計算する．

2.3 外力

最後に本節では，式 (2.1) の外力項に適用される外力のうち，提案手法で取り扱うものについてまとめる．本論文では，外力として浮力，重力および渦補正力を用いる．以下でそれぞれの外力について詳しく説明する．

浮力 $\mathbf{f}_{buo}$ は，格子点の温度と環境温度との差に比例し，以下の式で与えられる．

$$\mathbf{f}_{buo} = \kappa_b(T - T_{amb})\mathbf{v} \tag{2.14}$$

ここで，κ_b は浮力の係数，$\mathbf{v}$ は鉛直上方向の単位ベクトルである．

重力は，各格子点の密度に比例し，以下の式のようになる．

$$\mathbf{f}_{gra} = -gD\mathbf{v} \tag{2.15}$$

ここで，g は浮力の係数である．

渦補正力は，数値誤差により失われる流体の乱流成分を外力的に付加し，補うものであり，文献 [9] で提案された．この外力は，渦度 ω から計算される力で以下の式により定義される．

$$\mathbf{f}_{conf} = \epsilon(\mathbf{N} \times \omega) \tag{2.16}$$

ここで，ϵ は渦補正力の強さを決める係数，ω は $\omega = \nabla \times \mathbf{u}$ として表わされる．また，$\mathbf{N}$ は ω を用いて，$\mathbf{N} = \nabla|\omega|/|\nabla|\omega||$ となる．

第3章　爆発シミュレーションの制御

CG において流体解析を利用した煙や水などの動きを計算する研究は盛んに行われている．また，流体の自然な動きを保ったまま，ユーザの意図した形状となるよう流体の動きを制御する研究も行われており，ゲームや映画など幅広い分野への応用が期待されている．このような映像作品において，流体が特定の形状として表現される場合がある．しかし，シミュレーションにおいてパラメータを調整するだけでは，特定の形状として流体を表現することは非常に難しい．本章では，流体の中でも爆発のシミュレーションに注目し，流体力学に基づいた爆発のシミュレーションに対し，ユーザの指定した形状が生成されるようコントロールする手法を提案する．提案手法により，ユーザの意図したとおりの形状にコントロールされた爆発のシミュレーションが可能である．

3.1　研究目的

近年，コンピュータグラフィックス（CG）によって自然現象をシミュレーションする研究が盛んに行われている．その中でも水，煙，雲などの流体現象は映画やゲームなどの映像制作において重要な要素の一つであり，流体現象のシミュレーション手法が多く利用されている．このような流体現象は，流体力学に基づく数値シミュレーション（以下，流体シミュレーション）により写実的な表現が可能である [9, 25, 35]．そのため，流体シミュレーションを用いて効率良く，効果的に流体現象を表現する手法がこれまで数多く研究されている．しかし，これらの従来研究では写実的なシミュレーションに重点が置かれており，ユーザの意図を反映することは考慮されていなかった．

映画やゲームにおいて，煙や水，雲などが人や動物など現実にはない形状として表

現されていることがある．しかし，このようなアニメーションは手作業で制作されている場合が多く，多大な時間と労力が必要である．また流体シミュレーションを用いない場合，リアルな動きを与えることは困難である．そこで，流体シミュレーションによる自然な動きを保ちつつ，流体の動きを制御して様々な形状を表現するための手法がいくつか提案されている [6, 8, 34, 37]．しかしながら，これらの研究は煙や水，雲などを対象としており，爆発現象の制御に関する手法は存在しない．

映像制作において爆発現象を用いる際，実際の爆発を利用することは危険を伴うため，爆発のシミュレーションにより CG として製作される場合も多い．そのため，爆発を流体シミュレーションを用いて表現する手法がいくつか提案されている [10, 14, 17, 32]．これらの手法を用いることで，映画やゲームにおいて爆発やそれに類似した現象の写実的なアニメーションを生成できる．しかし，シミュレーションは多くのパラメータに依存しており，ユーザの意図した形状となるよう，これらのパラメータを調整することは極めて困難である．また，爆発発生時の急速な熱膨張による大きな速度や熱による浮力のため，従来の制御手法をそのまま用いて制御を行うことも難しい．

上記の問題を解決するため，本研究では，爆発シミュレーションに対して，ユーザの意図した形状に制御する手法を提案する．ここで，本手法における爆発の制御とは，爆発発生後に爆発が目標形状に制御された後，浮力により自然な形で上昇していくことを指している．このような制御を可能とするため，本手法では，爆発の初期強度分布の最適化および予測制御という 2 つの処理を爆発シミュレーションに対して適用する．本手法では，爆発の発生を圧力ではなく，ある円状の領域から放射上に広がる外力により擬似的に表現しており，この外力分布を爆発の初期強度分布とする．最適化処理において，ある指定された時間での爆発形状と目標となる形状との差分を最急降下法により最小化することで最適な爆発の強度を算出する．予測制御では，爆発形状の推移をモデル化し，それを用いてシミュレーション実行時に現在の爆発形状から将来の形状を予測する．そして，予測した形状とユーザが指定した目標となる形状との差をもとに外力を付加することで，爆発が目標形状に到達する前に詳細な形状の制御

を行う．これにより，爆発が目標を超えてから外力により発生源の方向へ戻る振動現象を回避することができる．また，本手法では制御手法を 2 次元の爆発を対象とし，3 次元の爆発は擬似的に表現する．提案手法により，ユーザの指定した形状に制御された爆発のアニメーションを生成できる．

3.2 関連研究

本研究は爆発のシミュレーションおよび流体解析のコントロールに関連しているため，以下でそれぞれに関する従来研究について議論する．

CG 分野において，流体解析に基づいて爆発をシミュレーションする手法は Yngve らによって初めて提案された [14]．この手法は，圧縮性の流体方程式を解くことで，爆発シミュレーションを行っているが，非常に時間がかかるという問題がある．Feldman らは，Navier-Stokes 方程式（N-S 方程式）の安定解法 [35] を応用して爆発のシミュレーションを行う方法を提案している [10]．この手法では，Gas model と Particulate model を構築し，3 次元における N-S 方程式による流体解析と燃焼物やすすを表す粒子との間に発生する相互作用を考慮することで，爆発をシミュレーションしている．Kang らは格子法と粒子法を組合せた爆発のシミュレーション方法を提案した [17]．しかしこれらの手法は，生成される爆発の形状が多くのパラメータに依存しているため，ユーザの意図した形状となるよう，適切なパラメータを決定することは極めて困難である．

流体現象のコントロールに関する研究として，キーフレームごとの煙の形状をコントロールする方法が Treuille らによって提案されている [37]．これは，シミュレーション空間に配置した格子点から外力を発生させることで，目的の形状を表す密度分布に近づける手法である．また，Fattal らは N-S 方程式の外力項に新たな二つの外力を追加することで，煙の形状変化アニメーションを生成する手法を提案した [8]．また，フィードバック制御により，水の形状変化を行う方法も提案されている [34]．Dobashi らは，リアルな積乱雲のシミュレーションに対し，ユーザが指定した形状に一致するようコ

ントロールする手法を提案した [6]．しかし，これらの手法は煙，水および雲を対象とした手法であり，爆発のシミュレーションを対象とした手法はいまだに存在しない．

本章では上述の問題を解決し，爆発の初期強度の最適化および予測制御により，爆発シミュレーションの制御を実現する手法を提案する．提案手法により，複雑なパラメータ設定を行わずに爆発のシミュレーションを制御することが可能である．

3.3 爆発のシミュレーション

爆発のシミュレーション方法について説明する．本手法の制御手法は 2 次元における爆発を対象としており，以下に述べる簡易モデルによりシミュレーションを行う．まず，2 次元のシミュレーション空間を $N_x \times N_y$ の格子に分割し，その内部に爆発を発生させる点（爆発源）を配置する（**図** 3.1 参照）．そして，分割した各格子点に速度 $\mathbf{u}$ および密度 D，温度 T を割り付け，それらの値の時間変化を計算する．密度は爆発により飛散する燃焼物を表すものとする（図 3.1 を参照）．爆発の発生のために爆発源から放射状に向かう大きな外力と密度を短時間だけ与える．この外力により，瞬間的に大きな速度が発生するので，爆発が発生する様子を擬似的に表現できる．なお以下では，爆発源に与える外力の大きさを爆発の強度とする．その後，シミュレーション空間内の速度場，密度場および温度場の時間変化を第 2 章の方法を用いて流体解析を行うことで爆発をシミュレーションする．

次に，上述した 2 次元の爆発のシミュレーション方法を用いて 3 次元の爆発を生成する手法について説明する．3 次元への拡張は Rasmussen らの手法 [32] を利用する．この手法では複数の 2 次元シミュレーションを組み合わせて 3 次元の流れ場を補間により生成する．以下で詳細について説明する．

3 次元のシミュレーション空間にユーザが指定した数の 2 次元平面を回転対称に配置する (図 3.2 参照)．そして，各平面内において，2 次元での爆発シミュレーションを実行する．得られた結果からシミュレーション空間内の流れ場を補間により生成する (図 3.2 参照)．本手法では，単純に線形補間を利用した．次に，爆発源に多数の粒

子（パーティクル）を生成し，補間した 3 次元の流れ場に沿って移動することで 3 次元での爆発を生成する．また，パーティクルは速度と同様の方法によって補間された密度および温度を持つ．以上の処理により 2 次元の爆発を 3 次元へ拡張できる．

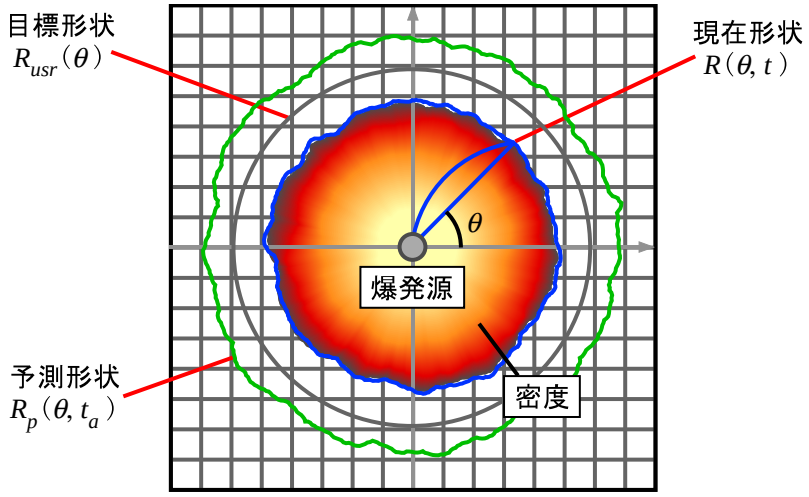


図 3.1: パラメータの定義

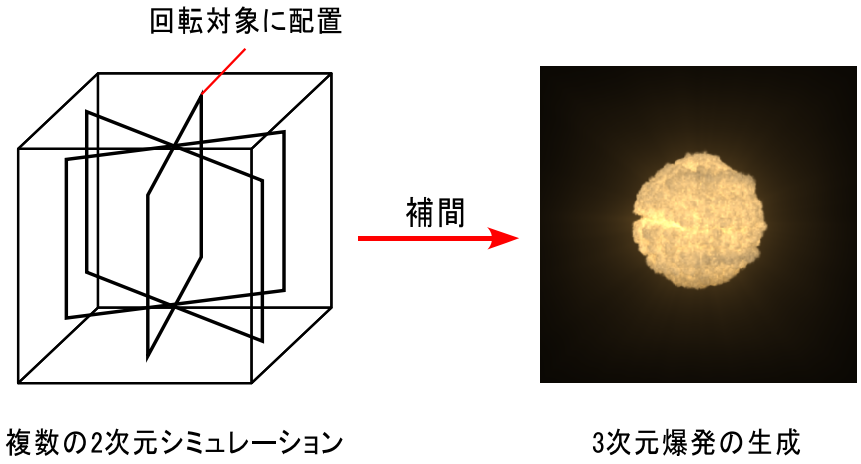


図 3.2: 3 次元爆発の生成

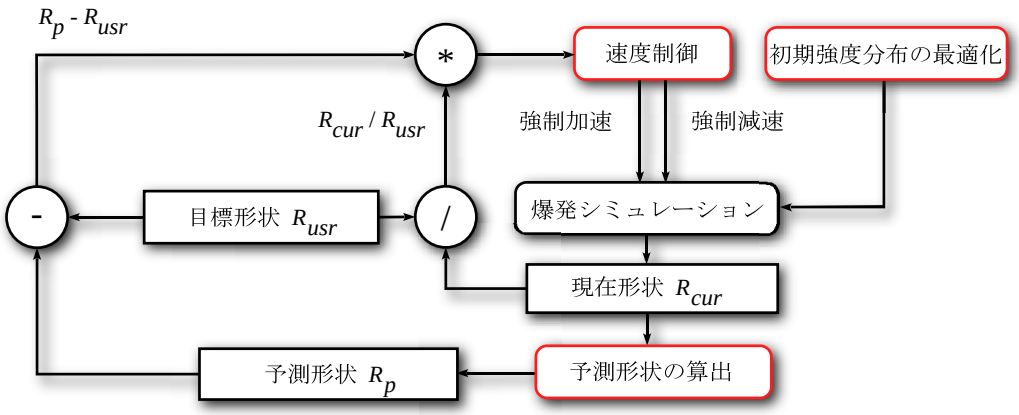


図 3.3: 処理の流れ

3.4 爆発の制御方法の概要

本手法における爆発の制御方法について説明する．以下，ユーザにより指定された目標形状を $R_{usr}(\theta)$ と表す（図 3.1 参照）．シミュレーションにより生成される爆発の時刻 t における形状は，爆発源から角度 θ 方向について，密度が存在する最遠方の格子点までの距離 $R(\theta, t)$ として表す（図 3.1 参照）．また，ユーザは目標形状とあわせて爆発の目標への到達時間 t_a を指定する．この時刻 t_a において二つの形状が一致するようシミュレーションを制御する．またこのとき，爆発源は目標形状内部にユーザが任意に配置する．**図** 3.3 に提案法による処理の流れを示す．爆発の制御は，初期強度分布の最適化と予測制御からなる．

初期強度分布の最適化は前処理として行い，与えられた目標形状に爆発が一致するような初期強度分布の探索を行う．ただし，本手法では，比較的簡易な手法を用いて最適解に近いものを推定する．そして，予測制御により更なる精度向上を図る．また，このとき浮力を考慮して探索を行うことで，浮力による影響を考慮しつつ目標形状に近づくような初期強度分布の探索が可能である．

初期強度分布を決定した後，爆発のシミュレーションを開始し，予測制御により爆発をより高精度に目標形状に一致させる．具体的には，現時刻から一定時間 Δt 後の予測形状 $R_p(\theta, t + \Delta t)$ を算出し（図 3.1 参照），目標形状との差から速度場の制御を

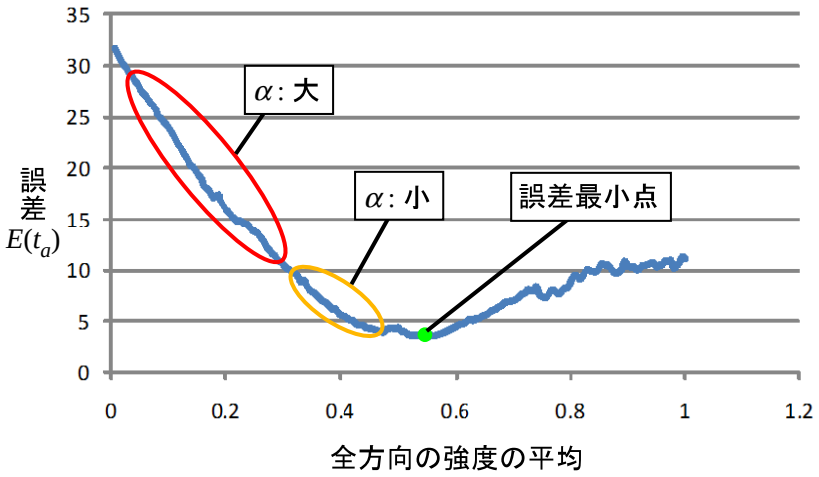


図 3.4: 強度分布と誤差の関係

行う．以下で各処理について説明する．

3.5 初期強度分布の最適化

さまざまな初期強度分布について繰り返しシミュレーションを行い，より目標形状に近い爆発が生成するような初期強度分布を探索する．すなわち，時刻 t_a における爆発形状と目標形状との誤差に応じて，強度分布を更新しながらシミュレーションを繰り返す．そして，誤差が最小となる強度分布を最終的な爆発の初期強度として設定する．本手法では，最急降下法を参考として最適値の算出を行うが，パラメータ数が多く計算時間がかかるため，より簡易な方法により最適値の算出を行う．以下で詳細について説明する．

3.5.1 最適な強度分布の探索

時刻 t_a までシミュレーションを行ったときの爆発形状と目標との誤差の値に応じて次式により強度分布を更新する．

$$I_{opt}(\theta, n+1) = I_{opt}(\theta, n) + \alpha(R_{usr}(\theta) - R(\theta, t_a)) \tag{3.1}$$

ここで，$I_{opt}(\theta, n)$ は n 回目のシミュレーションにおける角度 θ 方向の強度，α は強度分布の更新の度合いを調整するための係数である．$I_{opt}(\theta, n)$ の初期値 $I_{opt}(\theta, 0)$ は以下の式により設定する．

$$I_{opt}(\theta, 0) = I_{usr} \cdot R_{usr}(\theta) \tag{3.2}$$

ここで，I_{usr} はユーザにより指定する係数である．式 (3.2) により，爆発をおおよそ目標形状に一致させるような強度を設定することができる．また α については，1 回のシミュレーションごとに最適な α を算出する．その方法については次節で詳しく説明する．一方，t_a における爆発形状と目標形状との誤差 $E(t_a)$ は以下の式に示すように全方向の誤差の平均として表す．

$$E(t_a) = \frac{1}{N} \sum_{n=0}^{N-1} |R_{usr}(\frac{2\pi n}{N}) - R(\frac{2\pi n}{N}, t_a)| \tag{3.3}$$

ここで N は角度 θ のサンプリング数であり，サンプリング間隔は $\frac{2\pi}{N}$ となる．この誤差 $E(t_a)$ が最小となるまで式 (3.1) による更新を繰り返し，最小となったときの強度分布を最終的な初期強度とする．**図** 3.4 に全方向の強度の平均と誤差 $E(t_a)$ との関係を表したグラフの一例を示す．このグラフにおける誤差最小点（図 3.4 の誤差最小点）を本節の処理により探索する．

3.5.2 強度分布探索の高速化

前節の式 (3.1) における係数 α の設定方法について説明する．上述したとおり α は強度分布の更新の度合いを調整する係数であるが，この値が大きいと強度分布が収束せずに発振する場合があり，小さいと強度分布の探索に時間がかかってしまう．そこで，1 回のシミュレーションごとに最適な α を探索する．この探索には直線探索を応用する．具体的には，α を数種類設定しすべての α において誤差の算出を行う．我々の実験では 2 種類の α のみで充分な結果が得られたため，α として 0.1 と 0.5 を使用した．そして，誤差が最も小さくなった α をその試行での最適な α とする．これを強度分布探索の各試行において行うことで，α を動的に変化させることができ，誤差が

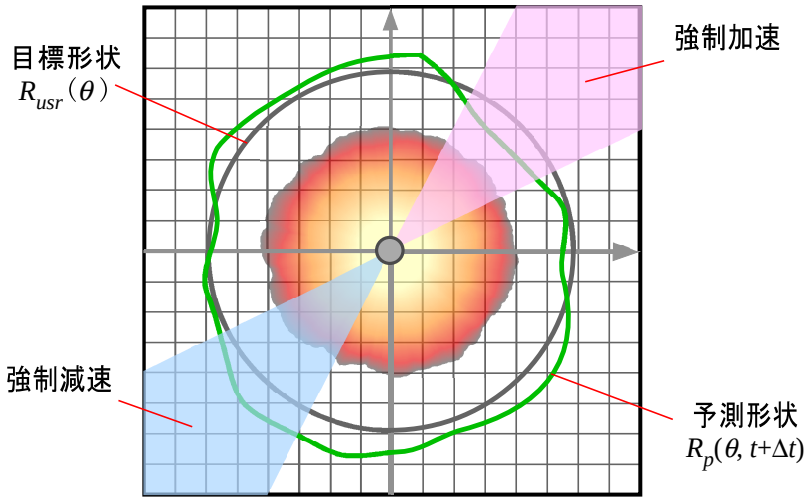


図 3.5: 速度場の制御

最小点から遠い場合は更新の度合いが大きくなるため，少ない試行回数で最小点付近まで達することができる（図 3.4 の α：大）．一方，最小点付近では更新の度合いが小さくなるため，より高精度に最小点の探索を行うことができる（図 3.4 の α：小）．この α の探索処理により，α を固定して最適化処理を行った場合に比べ，探索に要する時間を大幅に短縮できる．

3.6 予測制御

上述の初期強度分布の最適化処理により，目標形状に近い形状の爆発を生成することができる．さらに高精度に目標形状へ制御するために予測制御を行う．予測制御では，爆発の成長する速度から爆発の未来の形状を予測し，爆発の予測形状として算出する．そして，算出した予測形状と目標形状との差を求め，その差によって速度場の制御を行う．また，予測制御を時刻 t_a で停止させることで形状を制御しつつ自然なアニメーションを生成することができる．以下で各処理について説明する．

3.6.1 予測形状の算出

予測形状を算出するために，予測モデルを作成する．まず，3 節で述べた制御を行わない場合の爆発のシミュレーションを前処理として行っておく．この場合，半径が徐々に大きくなる円状の爆発が発生する．この円状爆発の成長速度（単位時間当たりの平均半径の変化率）を計測し，時間積分を行うことで，成長速度が $\mathbf{v}(t)$ であったときの一定時間 Δt 後の爆発半径を表す関数 $R_{ref}(\mathbf{v}(t), \Delta t)$ を以下のように構築する．

$$R_{ref}(\mathbf{v}(t), \Delta t) = \int_{t}^{t+\Delta t} \mathbf{v}(\tau) d\tau \tag{3.4}$$

この式の Δt を一定間隔で変化させたときに求められた値をグラフ化したものは，それぞれ以下の式で表される対数関数によって精度よく近似できる．

$$R_{ref}(\mathbf{v}(t), \Delta t) = a \cdot \ln(\mathbf{v}(t)) + b \tag{3.5}$$

ここで a，b はともに係数であり，Δt の値に応じて変化する．本手法では，a および b を Δt に関する三次関数により近似する．そして，式 (3.5) で表される $R_{ref}(\mathbf{v}(t), \Delta t)$ および現時刻における爆発形状 $R(\theta, t)$ を用いて，予測形状 $R_p(\theta, t + \Delta t)$ を次式により求める．

$$R_p(\theta, t + \Delta t) = R_{ref}(\mathbf{v}(\theta, t), \Delta t) + R(\theta, t) \tag{3.6}$$

ここで，$\mathbf{v}(\theta, t)$ は現時刻での θ 方向の爆発の成長速度を表す．本研究では，角度 θ に対して現在形状の曲線が存在する格子点の速度をその角度における爆発の成長速度とする．また，Δt は到達時間 t_a と現在の時刻 t との差から $\Delta t = t_a - t$ のように設定する．これを式 (3.6) に適用することで時刻 t_a に爆発が目標形状に達するよう制御できる．また，時刻 t_a で予測制御を停止させることで形状を制御しつつ，浮力への対応が可能である．上記の手法により得られた予測形状を目標形状と比較する．予測形状が目標形状を超えている場合は強制減速の処理を行い，予測が目標に達しない場合は強制加速の処理を行う．

3.6.2 強制減速

強制減速処理では，予測形状が目標形状を超えている方向に属している格子点（**図** 3.5 強制減速部分）に存在する速度を減少させる．これにより，爆発の成長を抑制できるので，爆発が目標を超えてしまうことを防ぐ．減少値 $\mathbf{u}_d(\theta, t)$ は以下のように予測形状と目標形状との差に比例させる．

$$\mathbf{u}_d(\theta, t) = \alpha_d(R_{usr}(\theta) - R_p(\theta, t + \Delta t))\mathbf{n}(\theta) \tag{3.7}$$

ここで，α_d は係数であり，$\mathbf{n}(\theta)$ は角度 θ 方向における爆発源から放射方向に向かう単位ベクトルである．

3.6.3 強制加速

予測が目標形状に達していない場合，この強制加速の処理を行う．この処理は予測形状が目標形状に達していない方向に属している格子点（図 3.5 強制加速部分）に存在する速度を増加させる．これにより，爆発の成長を促進できるので，爆発が目標に達しないということを防ぐ．増加値 $\mathbf{u}_a(\theta, t)$ についても以下のように予測と目標との差に比例させる．

$$\mathbf{u}_a(\theta, t) = \alpha_a(R_{usr}(\theta) - R_p(\theta, t + \Delta t))\mathbf{n}(\theta) \tag{3.8}$$

ここで，α_a は係数である．

以上の処理により爆発をユーザの意図したとおりの形状に制御することができる．

3.7 3 次元化

本節では，上述した 2 次元の爆発のコントロール手法を用いて，3 次元の爆発シミュレーションを制御する手法について説明する．3 次元への拡張は Rasmussen らの手法 [32] を利用する．この手法では，複数の 2 次元シミュレーションを組合せて 3 次元の流れ場を補間により生成する．そのため，前節までに述べた 2 次元の爆発における制御手法をそのまま用いることができる．詳細については文献を参照していただきたい．

3.8 実験結果

本節では，提案手法を用いて，爆発のコントロールシミュレーションを行った結果を示す．実験環境は，CPU が Intel Core2Quad Q9400（メモリ 4GB），GPU が NVIDIA Geforce GTX 295 となっている．

爆発の表示方法について説明する．まず，2 次元における爆発では，各格子点に割りつけられている密度の値に応じて各格子に色付けを行うことで爆発を表示する．具体的には，各格子の周囲 4 点の格子点について割りつけられている密度の値に応じて色を設定し，4 点の平均の色をその格子の色として表示する．3 次元ではシミュレーション空間内に 3 次元の格子（以下，ボクセル）を配置し，各ボクセルの内部に存在するパーティクルの温度および密度をそのボクセルのボリュームデータとして変換する．そして，そのデータをもとにボリュームレンダリングを行う．以下に示す実験例では，コントロールシミュレーションに用いる各パラメータを表 3.1 に示す値に設定した．また，各実験での 2 次元シミュレーションにおけるシミュレーション空間の分割数，1 フレームのシミュレーション時間，強度分布探索に要した時間は表 3.2 に，3 次元の結果における補間結果生成にかかる時間，使用した 2 次元平面の数，3 次元のシミュレーション空間の分割数は表 3.3 に示したとおりである．

表 3.1: 実験に用いた制御パラメータ数値

	パラメータ名	数値
dt	タイムステップ	0.1
dh	格子幅	1.0
ν	動粘性係数	1.0×10^{-9}
κ_b	浮力の係数	1.3
T_{amb}	環境温度	0.0
c_r	冷却定数	0.25
c_k	熱伝導率	1.0×10^{-8}
N	角度のサンプリング数	120
I_{usr}	初期分布係数	1.0
α_{adj}	予測関数補正係数	1.0
α_a	強制加速係数	3.0
α_d	強制減速係数	3.0
t_a	到達時間	4.0

表 3.2: 各実験例での 2 次元シミュレーションにおける計算時間・格子数

	シミュレーション	強度分布の探索	格子数
実験例 1(図 3.6)	0.08 秒/フレーム	35.4 秒	120×120
実験例 2(図 3.7)	0.08 秒/フレーム	39.1 秒	120×120
実験例 3(図 3.8)	0.04 秒/フレーム	平均 23.5 秒	120×120
実験例 4(図 3.9)	0.04 秒/フレーム	平均 27.8 秒	120×120
実験例 5(図 3.10, 3.11)	0.05 秒/フレーム	平均 26.4 秒	120×120
実験例 6(図 3.12)	0.08 秒/フレーム	−	120×120

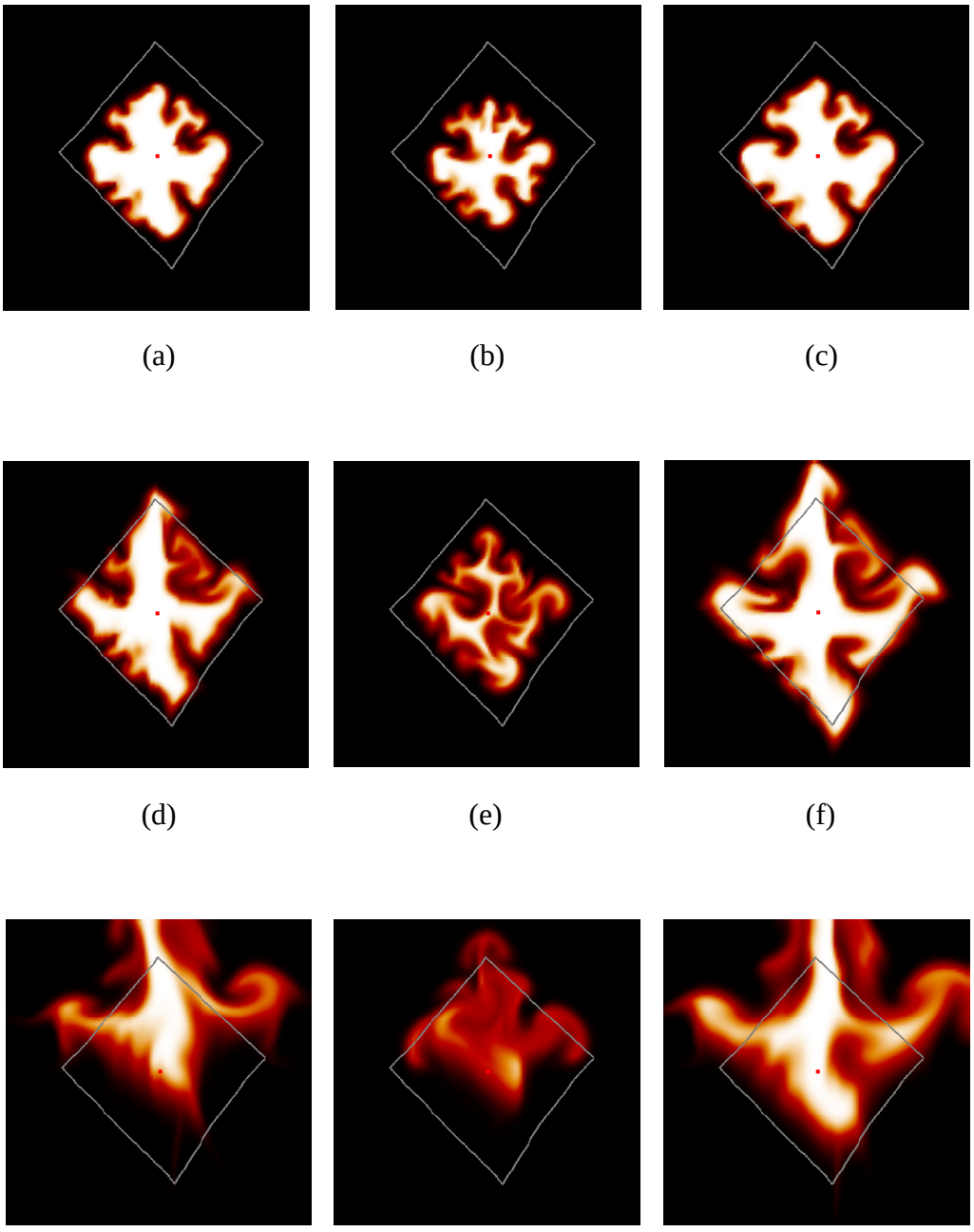
表 3.3: 3 次元の実験例での計算時間・格子数

	補間結果の生成	2 次元平面数	格子数
実験例 3(図 3.8)	1.7 秒/フレーム	3	$128 \times 128 \times 128$
実験例 4(図 3.9)	1.9 秒/フレーム	3	$128 \times 128 \times 128$
実験例 5(図 3.10, 3.11)	2.9 秒/フレーム	15	$128 \times 128 \times 128$

まず，提案手法の有効性を示すために，提案手法と従来手法を 2 次元シミュレーションにおいて比較した実験例について図 3.6 と図 3.7 に示す．従来手法には Fattal らの手法 [8] で用いられている外力による制御と Shi らの手法 [34] や Dobashi らの手法 [6] で利用されているフィードバック制御の 2 つの手法を利用し，爆発のシミュレーションと組み合わせた実験を行った．ただし，Fattal らの手法はそのまま適用するが，フィードバック制御に関しては目標形状と爆発形状との差を本手法における強制加速・強制減速の処理へフィードバックすることで制御を行った．図 3.6 は目標形状としてダイヤ型を，図 3.7 はハート型を指定した結果である．目標形状はそれぞれ図中に灰色の曲線として示す．図 3.6(a)(b)(c) は目的形状に達する前の結果であり，図 3.6(d)(e)(f) は指定した爆発の到達時間に目標形状へ達した結果である．また，図 3.6(g)(h)(i) は制御後，浮力により上昇している結果である．中央の画像が Fatal らの手法による結果であり，右側の画像がフィードバック制御による結果，左側の画像が提案手法による結果である．比較結果からわかるように，Fattal らの手法では，目標形状外部からの外力による制御が大きく働くため到達時刻での結果（図 3.6(e)）において目標形状に達しておらず，また浮力による上昇を外力が妨げており爆発が目標形状付近に停滞してしまっている．一方，フィードバック制御では到達時刻において目標形状を超えてしまっている．これは，目標形状と現在形状との差をフィードバックしているため，必要以上の外力が加わったことが原因である．しかし提案手法では，到達時刻において目標通りに制御されており，制御後も浮力により自然に上昇しているのがわかる．

次に，ハート型の目標形状を指定したときの結果について議論する．ダイヤ型を指定したときと同じく，図 3.7(a)(b)(c) は目的形状に達する前の結果であり，図 3.7(d)(e)(f) は指定した爆発の到達時間に目標形状へ達した結果である．また，図 3.7(g)(h)(i) は制御後，浮力により上昇している結果である．中央の画像が Fatal らの手法による結果であり，右側の画像がフィードバック制御による結果，左側の画像が提案手法による結果である．比較結果もダイヤ型の時と同様に，Fattal らの手法では，到達時刻において目標形状に達しておらず，また浮力による上昇を妨げるため爆発が目標形状付近

に停滞している．フィードバック制御においてもダイヤ型の時と同様，到達時刻において目標形状を超えてしまっている．しかし提案手法では，到達時刻において目標通りに制御されており，制御後も浮力により自然に上昇する．

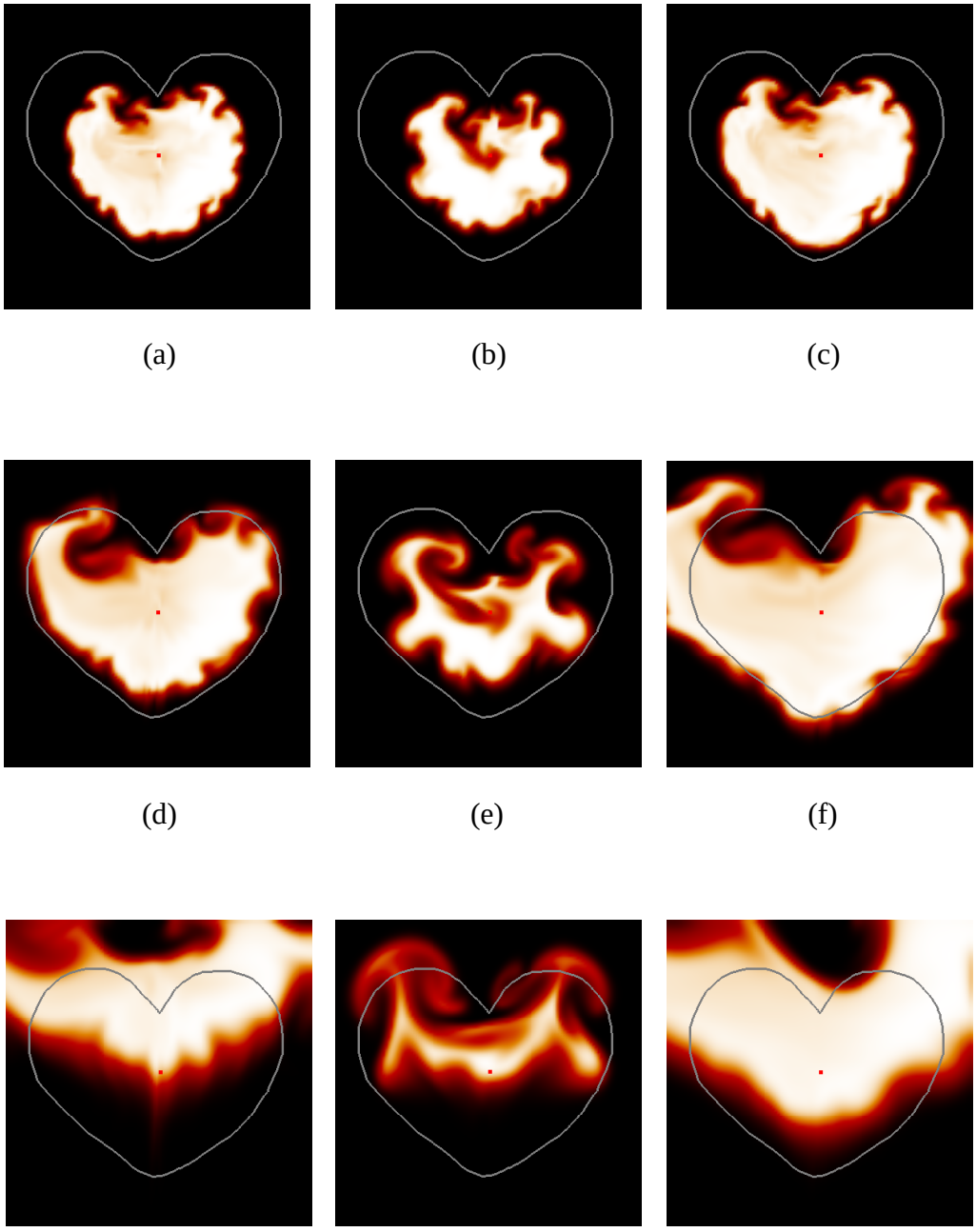


(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

図 3.6: 従来手法と提案手法の比較結果（目標形状：ダイヤ型）



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

図 3.7: 従来手法と提案手法の比較結果（目標形状：ハート型）

次に，提案手法を用いた適用例を図 3.8 と図 3.9，図 3.10 に示す．図 3.8 と図 3.9 では，球を目標形状として指定しているがその大きさが異なる結果である．それぞれの結果とも (a) が目標となる 3 次元形状を示している．また，(b)，(c)，(d)，(e)，(f) はそれぞれ 15step 目，40step 目，70step 目，100step 目，150step 目の結果を表している．どちらの結果とも指定した到達時間における (c) の結果において目標通りの形状が得られているのがわかる．さらに，それ以降の結果において爆発が自然に上昇していく．また，この 2 つの結果のように爆発の大きさを 3 次元形状で指定することで爆発の大きさを簡単に変えることが可能である．これをシミュレーションのパラメータを調整するだけで行おうとする場合，パラメータ調整とシミュレーションを繰り返し行わなければならず時間と労力を要する．

図 3.10 と図 3.11 は，目標形状としてドクロ形状を指定した場合の適用例である．図 3.10(a) が目標形状を表しており，(b)，(c)，(d) がそれぞれ 15step 目，40step 目，100step 目の結果を表している．この結果も球を指定した適用例と同様，到達時刻における (c) の結果において目標形状どおりの結果となっており，また目標へ到達後，浮力の影響により自然に上昇していく．提案手法を用いることで現実には存在しないような形状へ爆発をコントロールすることが可能である．また，図 3.11 は図 3.10 のドクロ形状の爆発を実写の画像と合成し，アニメーションを生成した例である．(a)，(b)，(c)，(d) はそれぞれ 15step 目，40step 目，70step 目，100step 目を表している．この例の場合，シミュレーション空間を空の部分に配置し，目標形状をドクロ型に設定して 3 次元の爆発の生成を行った．この例のように本手法による爆発の制御を用いることで，映画やゲームなどの映像に現実にはない形状の爆発アニメーションを簡単に作成することが可能である．

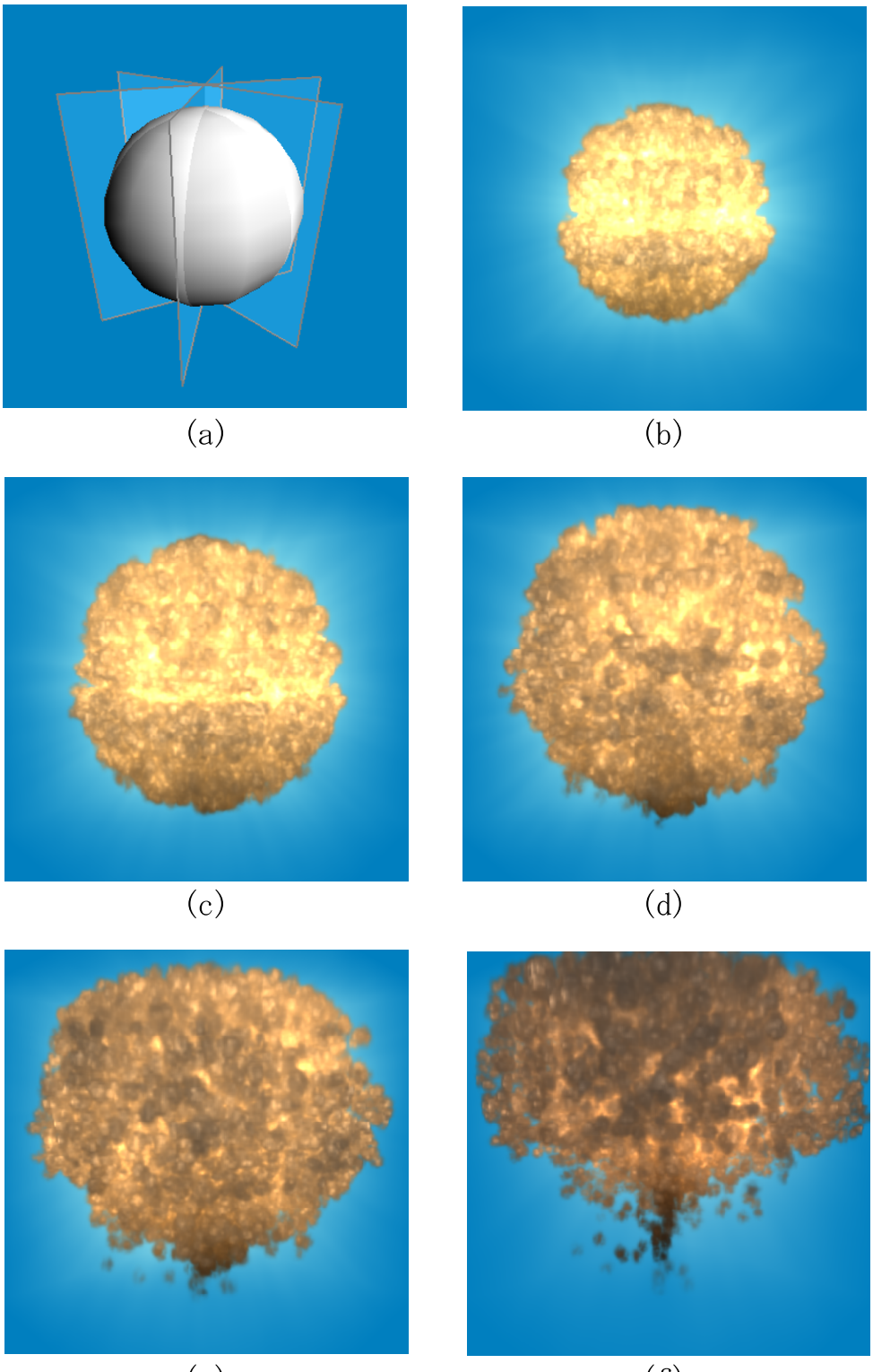


図 3.8: 単純な形状を指定した場合の適用例 1

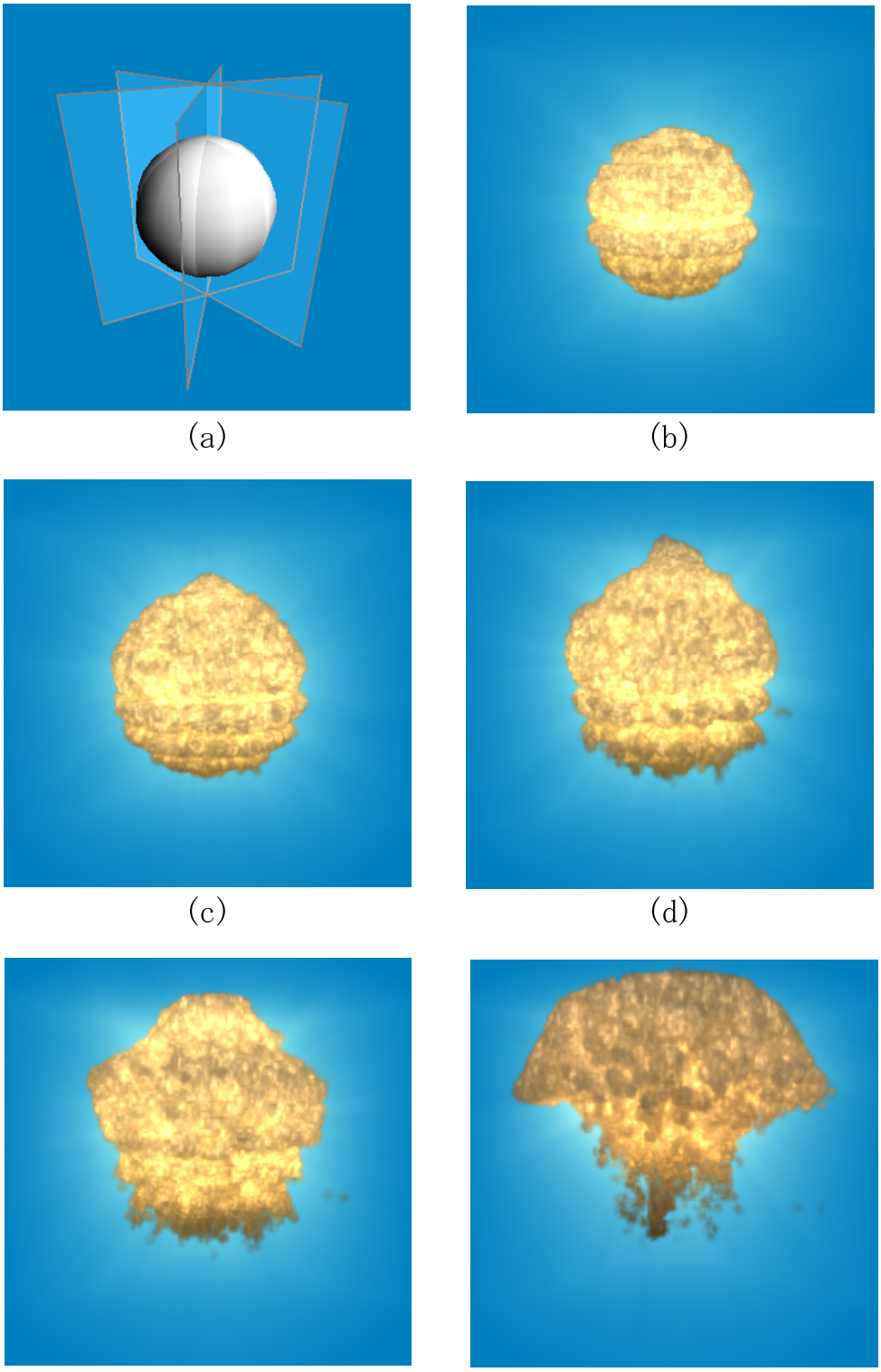


図 3.9: 単純な形状を指定した場合の適用例 2

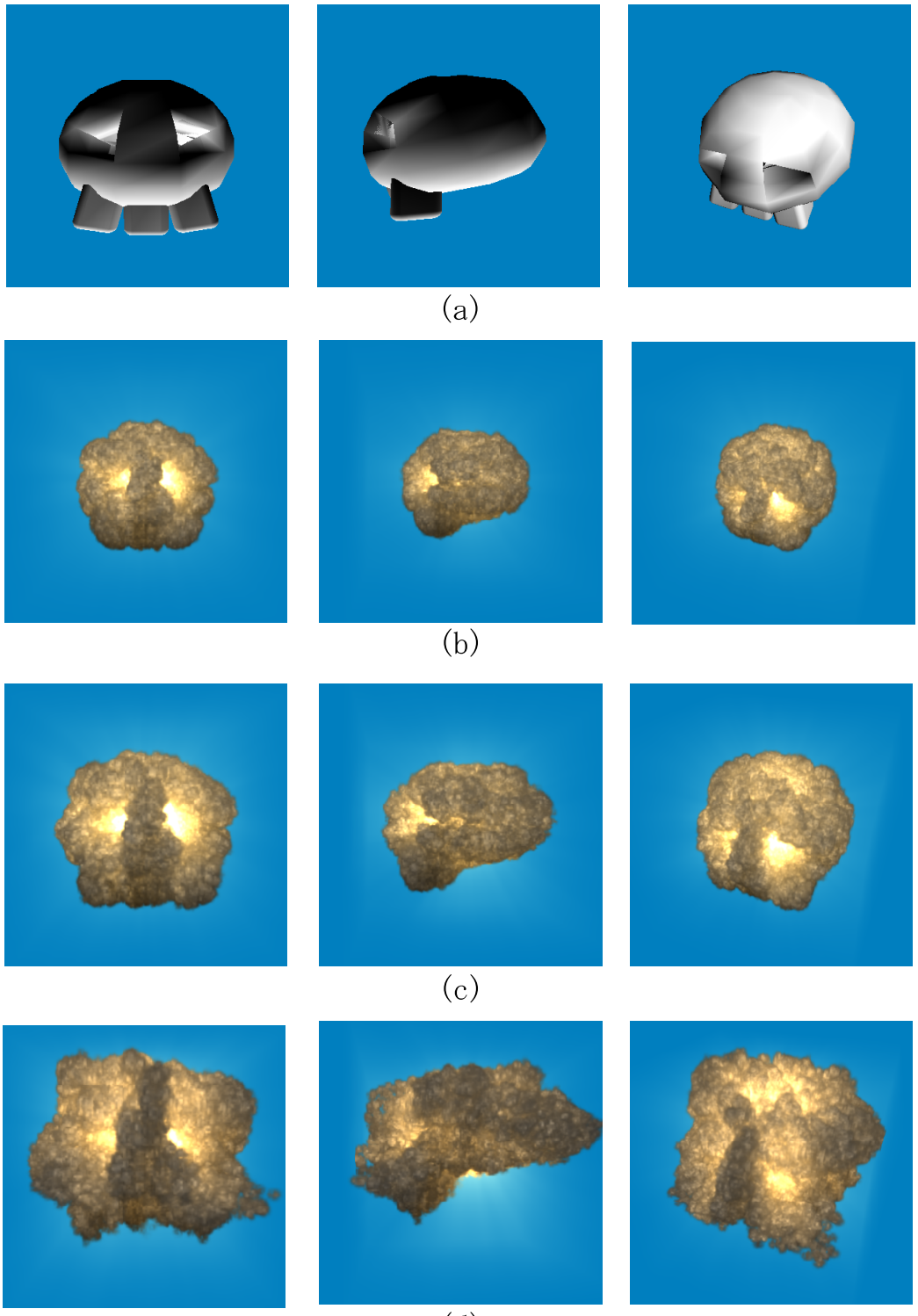


図 3.10: 複雑な形状を指定した場合の適用例

(a) (b)

(c) (d)

図 3.11: 実写との合成アニメーション例

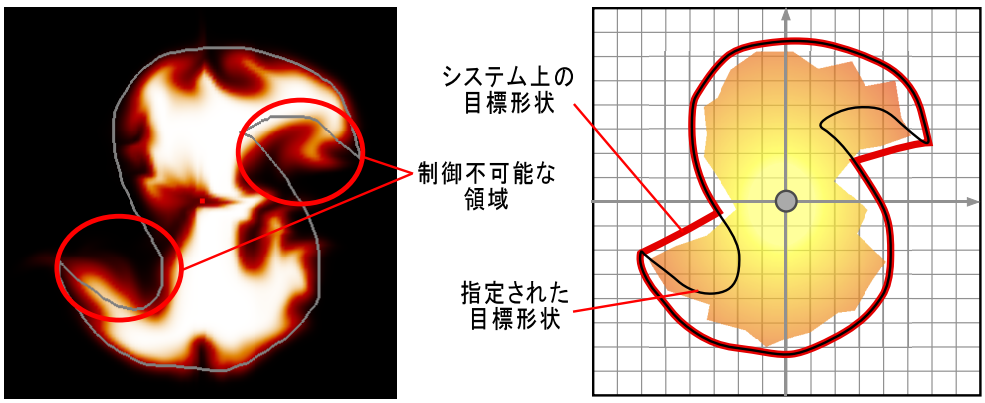


図 3.12: 失敗例

3.9 考察

本節では，3.8 節で示した実験例をもとに提案手法のコントロール方法の妥当性について検証する．

従来の流体解析のコントロール方法では，爆発の高速な動きに対応できないため目標形状どおりに制御することは難しい．このため，図 3.6 と図 3.7 の実験結果からもわかるとおり，目標形状どおりに制御できない場合や目標を超えてしまうといった問題が存在する．しかし，提案手法ではあらかじめ爆発の将来形状を予測し，予測結果に応じた制御を行うため爆発のような高速な動きをする流体に対して制御を実現できる．また，従来の爆発シミュレーションを利用する場合では多くのパラメータを調整しなければ意図した爆発を得ることは難しく，パラメータの調整だけでは得られない場合もある．しかし，本手法を利用することで意図した形状の爆発を煩雑なパラメータ設定なしに生成することができる．

本研究では，ユーザの希望する形状の爆発が生成されるよう爆発の動きをコントロールする方法を提案した．まず前処理として，指定された目標形状に最適な初期強度分布を設定することで大まかな爆発形状の制御を行い，さらにシミュレーション中に予測制御を行うことでより高精度に目標形状へ制御することが可能となった．しかし，提案手法では図 3.12 に示した S 字型など凸部分を含むような形状に対しては制御が不可能である．これは，本手法における目標形状の取得が爆発源からの放射方向に対して最も遠方の 1 点しか目標として設定できないためである．そのため，凸形状に対してはユーザが指定した目標形状とシステム上の目標形状が異なりユーザの意図した形状へ制御することができない．また本手法では，円（3 次元においては球）に近い形状に対しては制御精度がよいが，それ以外の形状に対しては制御精度がある程度落ちてしまう．これは，爆発源が円であることと，制御の方向が 1 次元（爆発源からの放射方向）のみであるということが原因としてあげられる．これら 2 つの問題点に対して，追跡用の粒子を用いることで解決できると考えられる．具体的には，爆発の速度場に沿って動く粒子をシミュレーション空間上に配置し，粒子の動きを追跡することで将

来の状態を予測し，さらに粒子を目標へ制御するような外力を制御力として用いることで凸形状のような複雑な形状に対しても制御が可能なのではないかと考えられる．

3.10 まとめと今後の課題

本手法では，初期強度分布を最適化することで，より自然な爆発のコントロールを可能とする手法について提案した．提案手法により，複雑なパラメータ設定を行わずに，ユーザの希望する形状へ爆発をコントロールすることが可能となった．また，シミュレーションに浮力を考慮したうえでの目標形状への制御も可能となった．

今後の課題としては，3 次元シミュレーションへの拡張があげられる．3 次元の場合，初期強度分布の最適化に要する計算コストが膨大になる可能性があるため，高速化を行う必要がある．高速化の手段として考えられる方法の一つに，流体シミュレーションの GPU 化があげられる．最適化処理における計算時間の大部分は，爆発シミュレーションの繰り返しが要因となっているため，流体シミュレーションの GPU 化により大幅な計算時間の削減が期待できる．また，他の流体の制御へ応用することなどがあげられる．

第4章　流体シミュレーションの超解像処理

流体シミュレーションを用いて，所望の流体映像を作成するには，シミュレーションパラメータを試行錯誤的に調整する必要がある．また，計算コストの高い流体シミュレーションを繰り返し実行しなければならず，映像が完成するまでの総時間は膨大なものとなってしまう．本章では，高速に計算可能な低品質のシミュレーションにより作成された結果を高品質な結果へ変換するための手法を提案する．これにより，ユーザは低品質なシミュレーションにより所望の結果を作成できるため，従来よりも効率的に映像作成を行うことができる．

4.1　研究目的

これまでも述べたように，数値流体解析を用いたシミュレーションは計算コストが非常に高いという問題がある．そのため，アニメータは所望の流体映像を得るために，計算コストの高いシミュレーションを繰り返しながら多くのパラメータを試行錯誤的に調整しなければならず，煩雑で時間のかかる作業となってしまう．

上記の問題を解決するために，低品質（低解像度）のシミュレーションを高品質（高解像度）化することで効率的に所望の流体映像を生成する手法の開発が行われている．この考え方を基に，高解像度シミュレーションを低解像度シミュレーションの結果を用いて制御する手法がいくつか提案されている [26][41]．しかし，これらの手法では最終的な結果のクオリティを確認するために，高解像度のシミュレーションを行わなければならない．一方，ノイズ関数と流体シミュレーションを組み合わせた手法も提案されており [19][29][33]，高解像度のシミュレーションを行わずに詳細な動きを含んだアニメーションを作成することができる．しかし，これらの手法により生成された結

果は，物理シミュレーションで得られた結果よりも写実性に欠ける結果となってしまう．本研究においても高解像度のシミュレーションを実行することなく，低解像度のアニメーションから高解像度の結果を作成する．本手法では，前計算により得られた流体シミュレーションの速度場を利用する．これにより，ノイズ関数を用いた手法と比較して写実性を向上させることができる．

本研究では，格子法による流体シミュレーションを対象としており，煙，炎，雲などといった気体現象のアニメーションの合成を目的としている．提案法は，流体アニメーションにおいて時間・空間的に異なる部分に類似したパターンの流れがあることに着目し，2 次元シミュレーションで作成した流れのパターンを用いて低解像度の速度場から高解像度の速度場を合成する．提案法は 2 つの処理により構成されている．まず，前処理として流体シミュレーションを実行し高解像度の速度場のデータベースを構築する．本手法では，データベースの構築に 2 次元の流体シミュレーションを用いる．次に，ランタイム処理では前計算した速度場データの線形和として低解像度の速度場を近似することで，高解像度での速度場を合成する．このとき，低解像度の 3 次元速度場を 2 次元の速度場データを用いて近似するため，3 次元速度場の x, y および z それぞれの成分ごとに近似処理を行う．また，低解像度の速度場をブロックに分割して処理を行うため，ブロック単位で並列に計算でき，GPU による並列処理を容易に実装可能である．

提案法によって高解像度化された流体の動きは，物理的に正しい動きを表しているわけではない．したがって，提案法による結果と物理シミュレーションによる結果は必ずしも一致しない．しかし，我々はこのことが重要な問題であるとは考えていない．なぜなら，本研究の目的は，視覚的にそれらしい映像を効率的に生成することであるためである．本手法では，提案法を用いて作成した炎，煙，雲のリアルな映像を示し，その目的が十分に達成できていることを示す．

4.2 関連研究

流体シミュレーションの計算コストを削減することを目的とした研究が数多く存在する [2][5][7][11][22][23]．しかし，高解像度でのシミュレーションの計算コストはいまだに高いのが現状である．また，所望の流体の動きを作成するためにはシミュレーションパラメータを試行錯誤的に調整しなければならず，多大な時間を要する．一方，2 次元の流体シミュレーションを用いて爆発 [32] や炎 [16] の高解像度のアニメーションを効果的に作成する手法が存在する．これらの手法は，高解像度の 3 次元流体シミュレーションと比較し，計算コストを大幅に削減できる．しかし，高解像度の 2 次元シミュレーションを複数回実行しなければならず，いまだ多くの計算時間を必要とする．上記の手法と比較し，本手法ではデータベースの構築の際に高解像度の 2 次元シミュレーションを 1 度だけ実行する．

近年，低解像度のシミュレーションから高解像度の結果を合成する手法がいくつか提案されている．Nielsen らは参照とする低解像度のシミュレーション結果に類似した高解像度の流体の動きを生成するための手法を提案した [26][27]．Yuan らは低解像度の流体シミュレーションから得た流れのパターンに従うよう高解像度シミュレーションを制御する手法を提案した [41]．これらの手法は，所望の流体の動きを含んだ高解像度の結果を生成できるが，計算コストの高い高解像度のシミュレーションを実行する必要がある．一方，低解像度の流体シミュレーションとノイズ関数を組み合わせることで，高解像度のシミュレーションを行わずに高解像度の結果を合成する手法が開発されている [19][29][33]．これらの手法では，詳細な動きを付加することができるが，高解像度のシミュレーション結果と比較して，写実性に欠ける結果となってしまう．

本手法は，ノイズ関数を用いる手法と高解像度シミュレーションを行う手法との間に位置する．高解像度の 2 次元シミュレーションにより速度場のデータベースを作成し，それを用いて低解像度シミュレーションから高解像度の結果を手続き的に生成する．Treuille らや Wicke らもデータベースを用いて流体シミュレーションを高速化する手法を提案した [38][39]．しかし，本研究の目的は低解像度シミュレーションを高解

像度の結果に変換することであり，シミュレーション自体の高速化ではない．さらに，それらの手法ではデータベース構築に 3 次元シミュレーションを用いるため，20〜30 時間という膨大な前計算を必要とする．本手法では，データベース構築を 2 次元シミュレーションで行うため，前計算時間を 1 時間程度にまで削減できる．

提案法と同様な考え方による研究として，高解像度の画像データベースを用いて低解像度の顔画像の超解像処理を行う研究が存在する [1]．しかし，この手法では必要なサンプル画像の数が多く，データベースの検索に時間がかかってしまう．また，検索を行う際の類似度計算の計算コストも高い．提案法では，計算の高速性とデータベース容量の低減を図るために，データベースの検索ではなく，データベースに格納された高解像度の速度場の線形和により与えられた低解像度の速度場を近似することで高解像度化を行う．また，ブロック分割を行い，かつ，速度の成分ごとに独立に処理を行うことで，より高速な処理を実現する．

4.3 気体現象のシミュレーション

本手法では，気体現象をシミュレーションするため，流体を非粘性，非圧縮と仮定する．気体の動きは，第 2 章の Navier-Stokes 方程式（以下，NS 方程式）を解くことで計算される．本手法では，この NS 方程式を文献 [5] を参考に GPU により数値解析する．

以下では上記の方程式を用いて煙，炎，雲をシミュレーションする方法を説明する．煙のシミュレーションでは，NS 方程式により得られた速度場にしたがって煙の密度を第 2 章の方法によって移流させる．次に，炎のシミュレーションでは，温度と浮力の影響を考慮する．これらに関しても，第 2 章の方法を用いて，解析を行う．次に，雲のシミュレーションでは，雲密度に加え水蒸気密度や相転移，潜熱の影響などを計算する必要がある．本手法では，Miyazaki らの手法 [31] を利用しているため，詳細についてはそちらを参照していただきたい．提案手法は，速度場にのみ適用され，高解像度における密度など他の物理量は合成された高解像度の速度場にしたがって移流させ

ることで動きを解析する．そのため，速度場の解析は，前計算および低解像度の 3 次元流体シミュレーションでのみ行う．

4.4 提案手法

提案手法の概要を図 4.1 に示す．前処理では，高解像度の 2 次元速度場のデータベースを構築する．次にランタイム処理では，3 次元流体シミュレーションにより低解像度の 3 次元の速度場を計算し，データベースを用いて高解像度の速度場に変換する．その後，密度や温度といったスカラ値を高解像度の速度場にしたがって移流させる．本手法では，2 次元のシミュレーション空間を uv 座標系，3 次元のシミュレーション空間を xyz 座標系で表す．以下で各処理について詳しく述べる．

4.4.1 データベースの構築

2 次元の流体シミュレーションを用いてデータベースを構築する．まず，シミュレーションを実行し，各タイムステップの速度場を一時的に保存する．その後，保存した各速度場をブロックに分割する．このとき，各ブロックが正方形となるよう分割し，その格子数は $n_b \times n_b$ とする．そして，ブロックに分割した速度場に対し主成分分析（PCA）を適用し，ブロック速度場の主成分を抽出する．抽出する主成分数はユーザにより任意に指定できるが，我々の実験では第 32 主成分までを利用すれば高品質な結果が得られることがわかったため，実験結果として示す例では主成分数を 32 としている．以降では，ブロック速度場の主成分を基本速度場と呼ぶこととする．算出した基本速度場をデータベースへ保存する．以降，基本速度場を $\mathbf{b}_i(i = 0, ..., m-1)$ として表す．ここで m は主成分の数を表す．

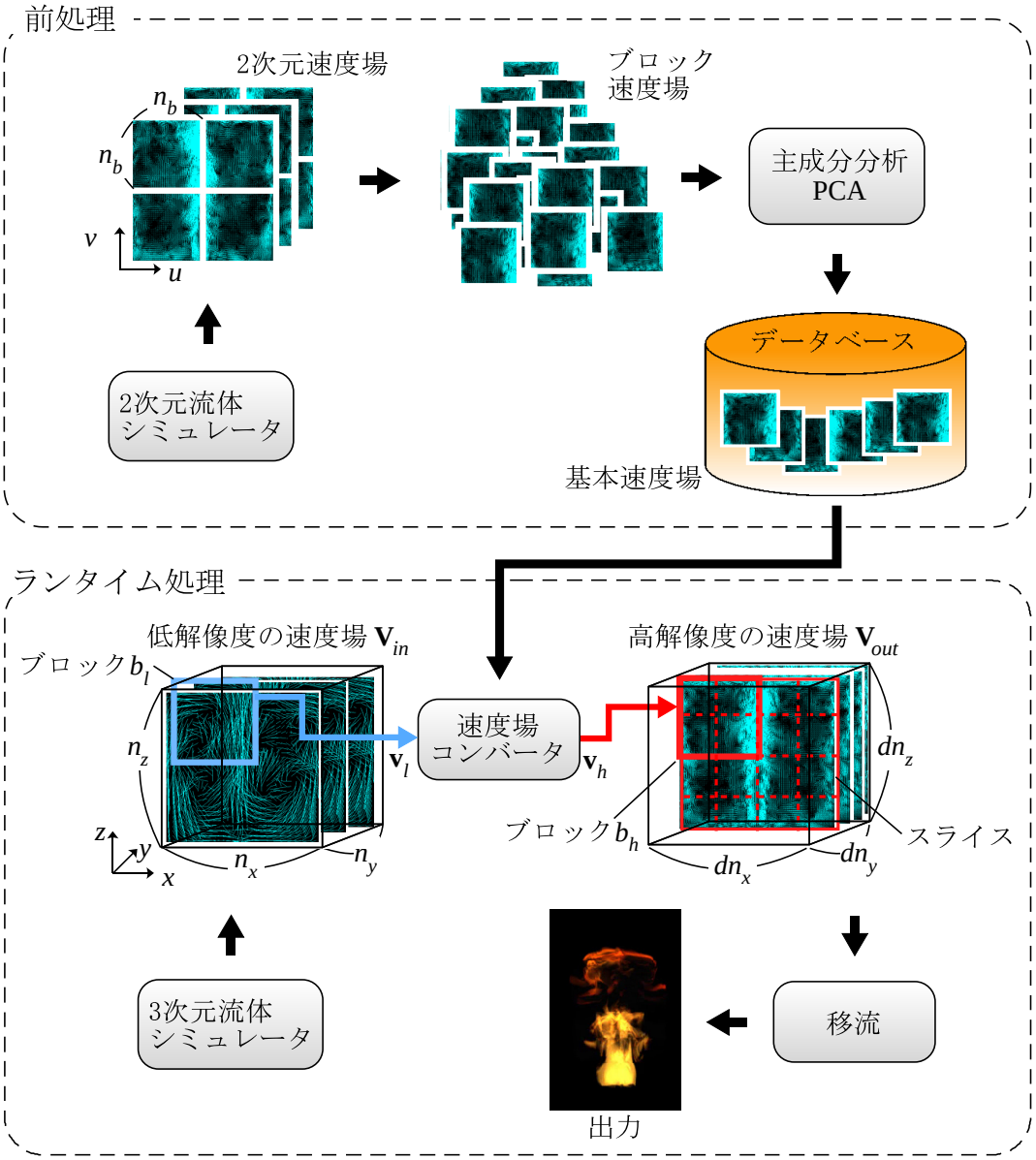

前処理
2次元速度場
n_b
n_b
v
u
ブロック
速度場
主成分分析
PCA
データベース
基本速度場
2次元流体
シミュレータ
ランタイム処理
低解像度の速度場 $\mathbf{V}_{in}$
ブロック b_l
n_z
$\mathbf{v}_l$
z
y
x
n_x
n_y
速度場
コンバータ
$\mathbf{v}_h$
高解像度の速度場 $\mathbf{V}_{out}$
dn_z
ブロック b_h
スライス
dn_x
dn_y
3次元流体
シミュレータ
移流
出力

図 4.1: 提案手法の概要

前計算に用いる 2 次元シミュレーションのパラメータ設定について説明する．本手法では，3 次元シミュレーションと同様のパラメータに設定する．例えば，3 次元において煙が上昇するシミュレーションを行う場合，2 次元でも同様の条件でシミュレーションする．また，解像度は生成したい高解像度の速度場と同様に設定する．乱流成分に関わるパラメータは，所望の乱流が得られるような値に設定する．我々の実験において，2 次元シミュレーションに所望の乱流が含まれていない場合，合成される結果に乱流を付加できないことがわかっている．上記のようにパラメータ等を設定し，データベースを構築する．

4.4.2 高解像度の 3 次元速度場の合成

ランタイム処理では，まず低解像度の 3 次元流体シミュレーションを実行する．そしてその各タイムステップにおいて，データベースに保存された基本速度場を用いて，3 次元の速度場を高解像度の速度場に変換する．本手法では，この変換を行う部分を速度場コンバータと呼ぶこととする．入力する 3 次元速度場を $\mathbf{V}_{in}(n_x, n_y, n_z)$ と表記する．n_x, n_y, n_z は格子数を表す．我々の目的は，入力の d 倍の解像度を持つ 3 次元速度場 $\mathbf{V}_{out}(dn_x, dn_y, dn_z)$ を合成することである．

以下で，合成処理の手順を説明する．まず，$\mathbf{V}_{out}$ の各スライス（図 4.1 : スライス）に対応する入力速度場 $\mathbf{V}_{in}$ のスライスを求める．このとき，y 軸方向の格子数の違いから，$\mathbf{V}_{out}$ のスライスに対応する $\mathbf{V}_{in}$ のスライスが存在しない場合がある．そのため，$\mathbf{V}_{in}$ における前後のスライスの線形補間により対応するスライスを生成する．次に，$\mathbf{V}_{out}$ および対応する $\mathbf{V}_{in}$ のスライスを同様のブロック配置となるよう小ブロック（図 4.1 : ブロック b_h，ブロック b_l）に分割する．このとき，ブロック間の不連続を低減するため，ブロックは重ねて配置する（図 4.1 : スライス上の点線）．本手法では，隣接するブロックと半分の大きさが重なるように配置し，重複部分には線形補間を適用する．そして，速度場コンバータにおいてブロック単位で速度場の高解像度化を行う．以降，高解像度のブロック速度場を $\mathbf{v}_h$ とし，入力となる低解像度のブロック速度場を

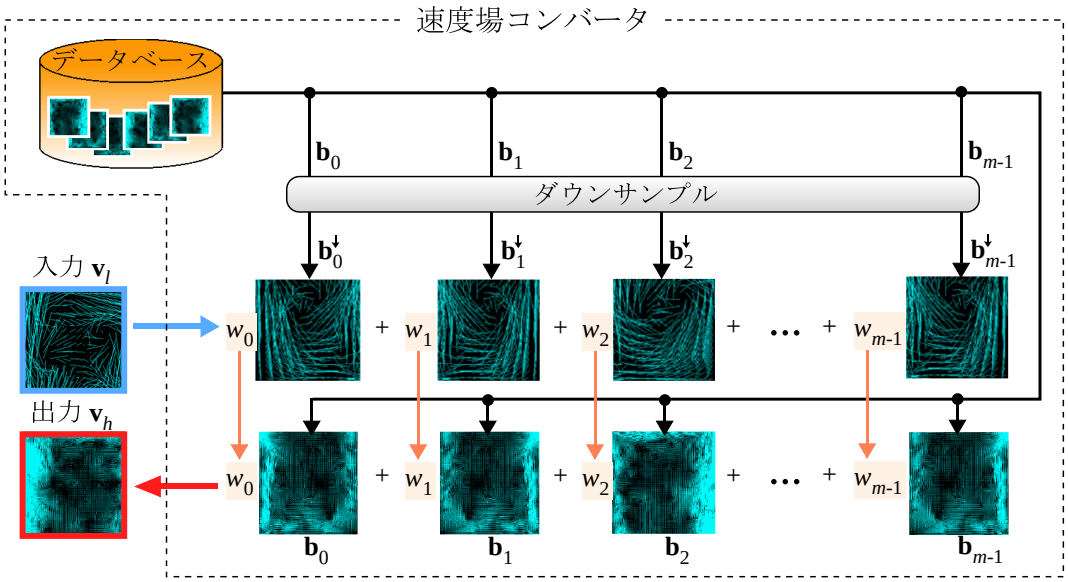


図 4.2: 速度場コンバータの詳細

$\mathbf{v}_l$ と表記する．以下では，$\mathbf{v}_l$ から $\mathbf{v}_h$ を合成する速度場コンバータの詳細を説明する．

図 4.2 に速度場コンバータの詳細を示す．まず，基本速度場 $\mathbf{b}_i$ を入力速度場 $\mathbf{v}_l$ の解像度へダウンサンプルし，その線形和で $\mathbf{v}_l$ を近似する．近似には，最小二乗法を用いる．そして近似処理により算出された重み $w_i (i = 0, ..., m-1)$ を元の基本速度場へ適用し，$\mathbf{v}_h$ を基本速度場の線形和として算出する．しかし，本手法では入力に 3 次元の速度場，データベースに 2 次元の基本速度場を用いているため，次元が異なる．そのため，入力速度場の xyz 成分それぞれに関して独立に上記の処理を適用する．このとき，基本速度場の水平方向成分（u 成分）を 3 次元速度場の水平方向成分（x，y 成分）に用いる．また，2 次元の垂直方向成分（v）を 3 次元の垂直方向成分（z）に用いる．

速度場コンバータにおける計算について説明する．まず，i 番目の基本速度場の u，v 成分をそれぞれ $\mathbf{b}_{i,u}$，$\mathbf{b}_{i,v}$ とする．同様に低解像度の 3 次元速度場の各成分を $\mathbf{v}_{l,x}$，$\mathbf{v}_{l,y}$，$\mathbf{v}_{l,z}$ と表記する．本手法では，以下の式を満たす重み w_i を算出することで $\mathbf{v}_l$ を

近似する．

$$\min_{w_{i,x}} ||\mathbf{v}_{l,x} - \sum_{i=0}^{m-1} w_{i,x}\mathbf{b}_{i,u}^{\downarrow}||^2 \tag{4.1}$$

$$\min_{w_{i,y}} ||\mathbf{v}_{l,y} - \sum_{i=0}^{m-1} w_{i,y}\mathbf{b}_{i,u}^{\downarrow}||^2 \tag{4.2}$$

$$\min_{w_{i,z}} ||\mathbf{v}_{l,z} - \sum_{i=0}^{m-1} w_{i,z}\mathbf{b}_{i,v}^{\downarrow}||^2 \tag{4.3}$$

ここで，$\mathbf{b}_i^{\downarrow}$ はダウンサンプルされた基本速度場を示し，m は基本速度場数を表す．上記の最小化問題はそれぞれ行列式として計算できる．計算方法は 3 成分とも同様であるため，以下では x 成分についてのみ説明する．式 (4.1) は次式の形に帰着できる．

$$\mathbf{A}_x\mathbf{w}_x = \mathbf{c}_x \tag{4.4}$$

ここで，$\mathbf{A}_x$ は $m \times m$ の行列であり，$\mathbf{w}_x$ と $\mathbf{c}_x$ は m 次の列ベクトルである．また，$\mathbf{w}_x$ の成分は重み $w_{i,x}$ となり，$\mathbf{A}_x$ の (i,j) 成分 $a_{ij,x}$ および $\mathbf{c}_x$ の i 番目の成分 $c_{i,x}$ は以下のようになる．

$$a_{ij,x} = \mathbf{b}_{i,u} \cdot \mathbf{b}_{j,u} \tag{4.5}$$

$$c_{i,x} = \mathbf{v}_{l,x} \cdot \mathbf{b}_{i,u} \tag{4.6}$$

式 (4.4) より重みは $\mathbf{c}_x$ に $\mathbf{A}_x$ の逆行列を乗算することで得られる．重みの計算後，ブロックの高解像度速度場 $\mathbf{v}_{h,x}$ は次式により合成される．

$$\mathbf{v}_{h,x} = \sum_{i=0}^{m-1} \alpha_i w_{i,x}\mathbf{b}_{i,u} \tag{4.7}$$

ここで，α_i は非負の制御係数であり，ユーザにより指定する．この係数を調整することで乱流の程度を制御することができる．本手法では，データベース構築の際に主成分分析を用いているため，乱流のような細かい動きの成分は基本速度場のより高次の主成分に含まれている．したがって高次の基本速度場に対する係数 α_i を大きくすることで乱流を強めることができる．

係数行列 $\mathbf{A}_x$ は入力となる速度場に依存していないため，その逆行列 $\mathbf{A}_x^{-1}$ は前もって計算しておくことが可能である．そのため，速度場の合成処理に含まれる計算は，行

列演算とベクトル演算となる．本手法では，これらの演算を GPU により実装している．また，各ブロックの計算は他のブロックとは完全に独立しているため，各ブロックでの計算を GPU で並列に処理することで，さらに高速化を図っている．

4.4.3 再帰的な合成

前節までで説明した提案手法では，高解像度化の度合い d が一定以上高いと，速度場が正しく近似されない．これは，基本速度場をダウンサンプルした際に，元の速度場とダウンサンプル後の速度場が大きく異なってしまうことが原因である．したがって，本手法で合成可能な解像度には上限があり，我々の実験では 4 倍を超えると不自然な結果となることが多く見られた．我々はこの問題に対し，本手法を再帰的に適用することで解決を図る．具体的には，まず入力速度場 $\mathbf{V}_i(n_x, n_y, n_z)$ を $\mathbf{V}_1(dn_x, dn_y, dn_z)$ に変換する．次に，$\mathbf{V}_1(dn_x, dn_y, dn_z)$ をさらに $\mathbf{V}_2(d^2n_x, d^2n_y, d^2n_z)$ へ変換する．ただし，再帰処理の各ステップにおいて所望の解像度でデータベースを再構築する必要がある．上記の処理を繰り返すことで，任意の解像度の速度場を生成できる．

4.4.4 密度および温度の移流

前節までの方法で生成した速度場を用いて，流体のスカラ量を移流させる．本手法では，煙の密度，炎の温度，雲の密度など最終的に可視化されるスカラ量のみ移流させる．移流の方法は以下のようになる．まず，スカラ量が格納されている高解像度の格子を用意する．そして，低解像度の 3 次元シミュレーションの各タイムステップにおいて新たに追加されたスカラ値分を高解像度の格子に加算し，高解像度の速度場にしたがって移流させる．煙では，ソースから煙の密度を追加する．炎の場合も，ソースから温度を追加する．しかし，雲に関しては明確なソースが存在しないため，水蒸気からの相転移により生成された雲密度をソースとして扱う．そのため，各格子点において相転移により生成された雲密度を高解像度の格子に加算する．さらに雲の場合，水滴から水蒸気への相転移により雲密度が減少する場合がある．この効果は対応する格子点の雲密度を減算することで考慮する．

4.5 実験結果

提案手法により生成した結果を図 4.3，4.4，4.5，4.6 に示す．実験環境は，CPU が Intel Core i7 2600K（メモリ 8GB），GPU が NVIDIA GeForce GTX 580 となっている．炎，煙のシミュレーションおよびレンダリングは GPU で計算を行っており，雲は CPU で計算を行っている．以下の実験結果では，基本速度場の格子数を 32×32 に設定し，主成分数を 32 としている．

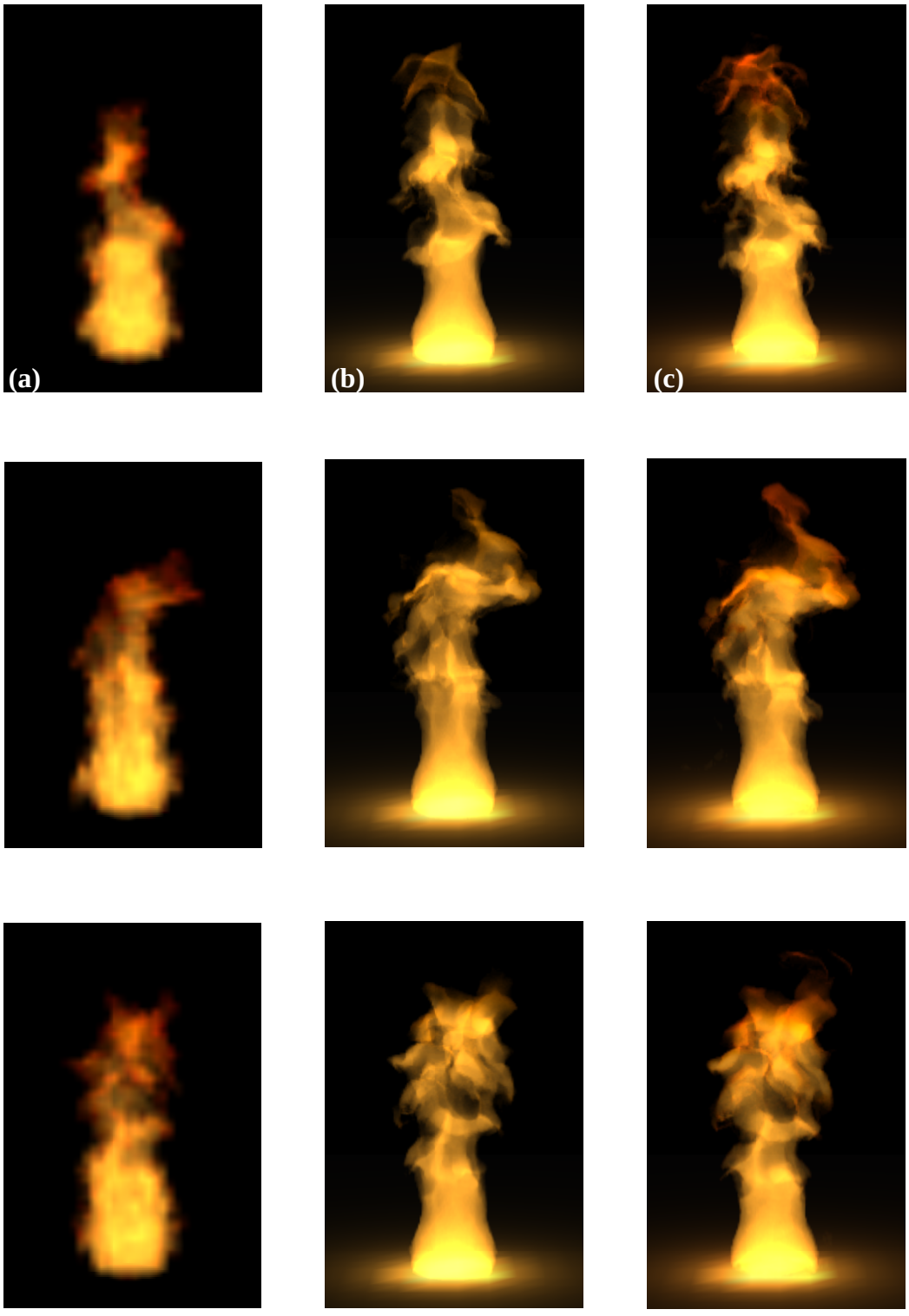
まず，炎の適用例により他の手法との比較を行う（図 4.3）．データベースの構築には格子数 128×256 の 2 次元シミュレーションを用いた．図 4.3(a) は低解像度シミュレーションの結果であり，格子数は $32 \times 32 \times 64$ である．図 4.3(b) は線形補間により $128 \times 128 \times 256$ の格子数に高解像度化した結果である．図 4.3(c) と (d) は図 4.3(a) から提案手法により合成した結果であり，図 4.3(d) は制御係数 α_i を調整し乱流を強めた結果である．図 4.3(e) は Kim らの手法 [19] を用いて生成した結果である．また，それぞれ上段の結果から一定時間経過後の結果を中段，下段に示した．図 4.3(b) では，提案法に比べ，かなり滑らかな炎となってしまっている．図 4.3(e) の結果では詳細な動きを付加できているが，ノイズ感の強い結果となってしまっている．一方，本手法による結果は，従来法に比べ写実的な結果となっているのがわかる．

図 4.4 は，風や動く物体の影響を考慮した結果である．これらの例では，単純に低解像度のシミュレーションにおいて，風や移動物体の影響を考慮して計算した後，提案法を適用している．データベース構築の際には風や移動物体の影響を考慮していないが，図 4.4 のように風や移動物体を考慮した写実的な映像の合成も可能である．図 4.5 には再帰的に提案手法を適用した例を示す．この例では，$32 \times 32 \times 64$ のシミュレーションに対して 3 回提案手法を適用し，格子数 $256 \times 256 \times 512$ の結果を合成した．このように提案法を繰り返し適用し，高い解像度の結果を作成することもできる．

次に，提案手法により煙と雲を合成した例を図 4.6 に示す．それぞれ白線から左側の部分が低解像度シミュレーションの結果であり，右側が合成結果である．煙の例では $64 \times 32 \times 32$ の速度場から $256 \times 128 \times 128$ の速度場を，雲の例では $96 \times 96 \times 60$

の速度場から $384 \times 384 \times 240$ の速度場をそれぞれ合成した．これらの例では，データベースの構築にそれぞれ煙と雲のシミュレーションを用いている．また，上段から下段にかけて一定時間経過後の結果をそれぞれ示している．結果からもわかるとおり，炎だけでなく煙や雲においても提案法を使用して高解像度の映像を作成することができ，気体現象のシミュレーションに対して汎用的に適用可能であることが確認できた．

最後に，表 4.1 および 4.2 に各シミュレーションの格子数と計算時間をまとめる．n_{2D} はデータベース構築に用いる 2 次元シミュレーションの格子数，n_{low} は低解像度の 3 次元シミュレーションの格子数，n_{high} は合成される高解像度の結果の格子数である．T_{pre} は前計算時間，T_{low} は低解像度シミュレーションの 1 ステップの計算時間，T_{synth} は速度場の合成とスカラ量の移流にかかる合計の計算時間，T_{high} は実際に高解像度でシミュレーションした場合の 1 ステップの計算時間を示す．雲に関しては，高解像度での水蒸気密度，雲密度，温度および速度を持つだけのメモリ量を確保できなかったため，T_{high} を省略している．表から高解像度でのシミュレーションに比べ，提案手法では約 3 倍高速に結果を生成できることがわかる．


(a)
(b)
(c)

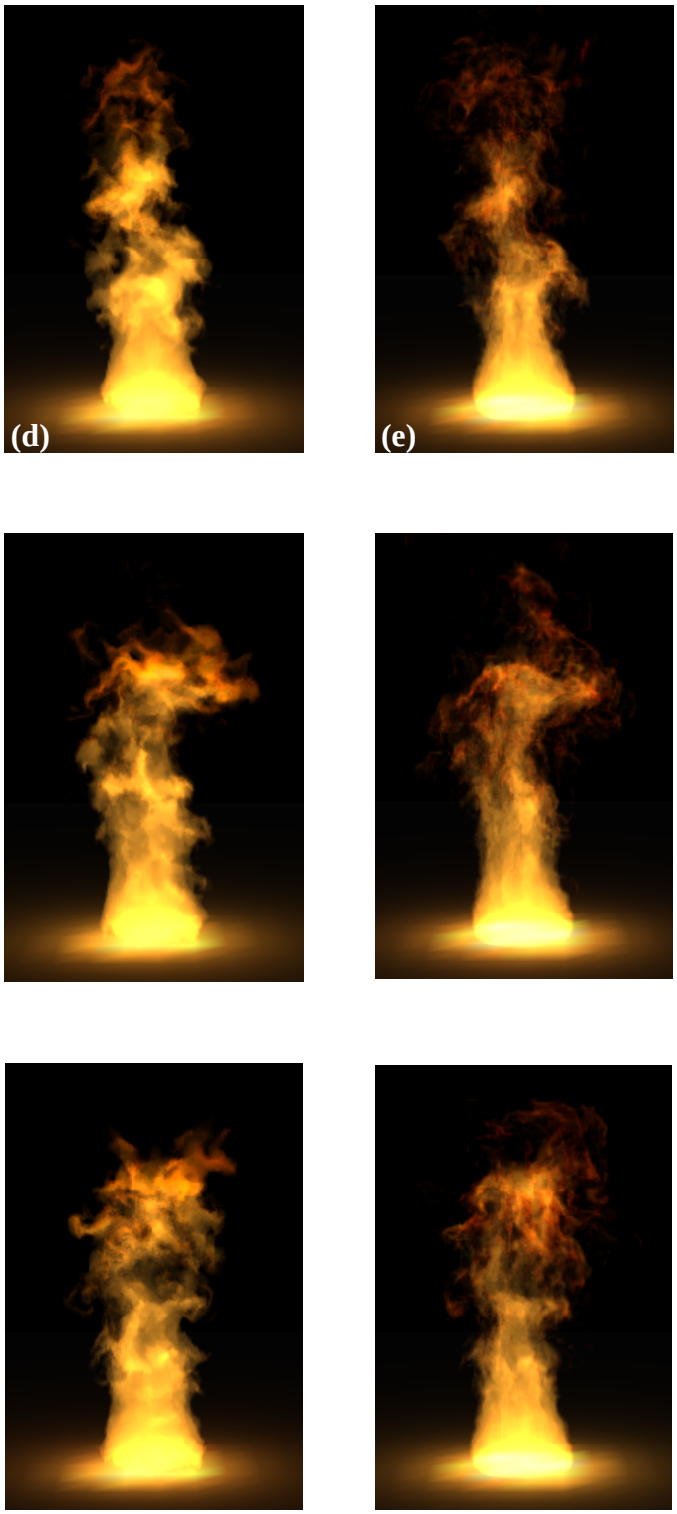


図 4.3: 炎による結果の比較

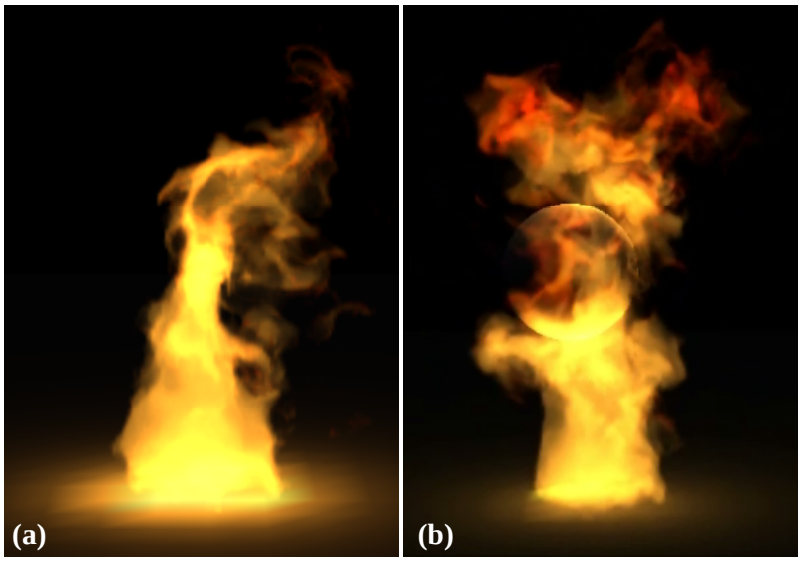
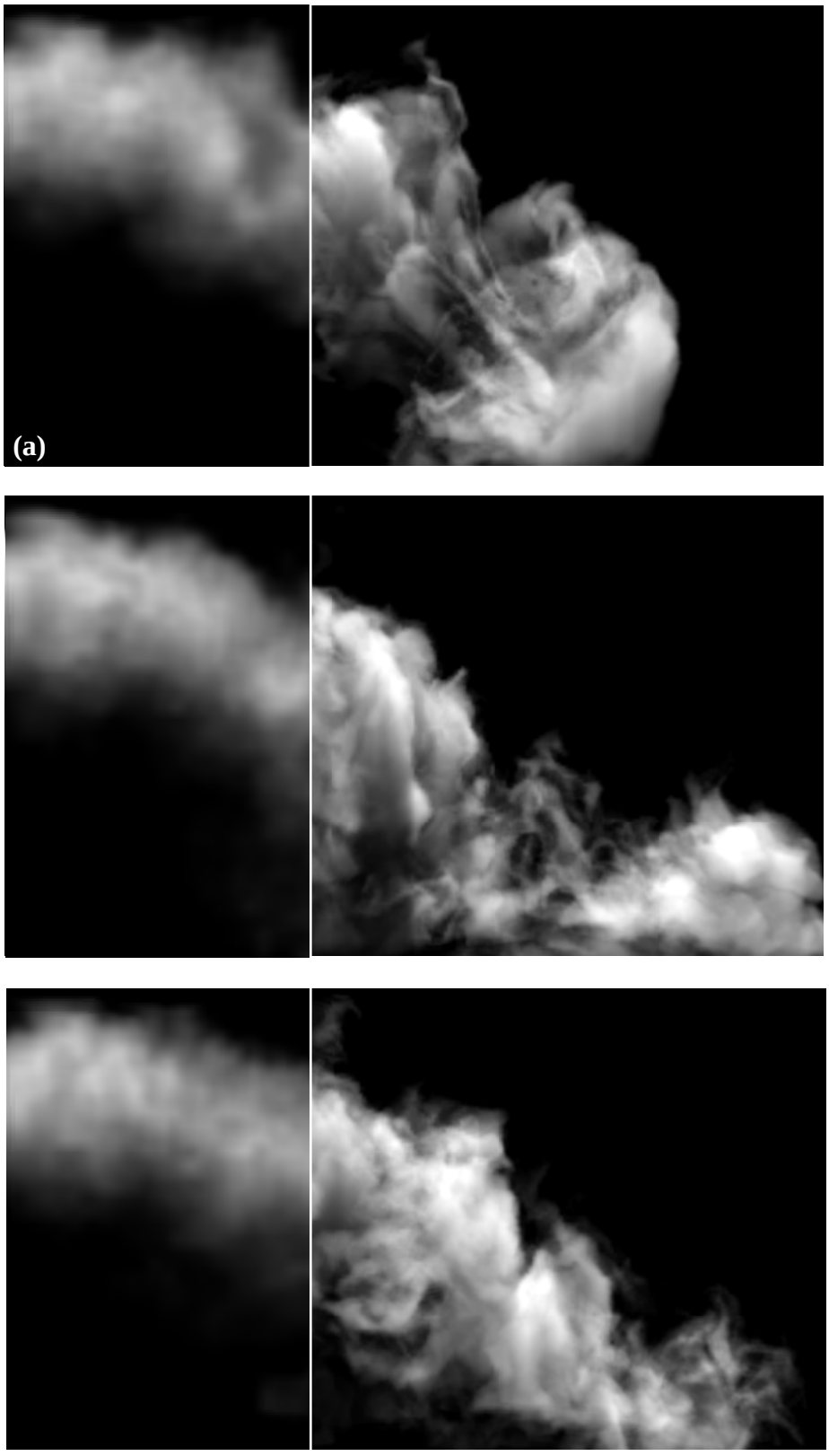


図 4.4: 風 (a) や障害物 (b) による効果

図 4.5: 再帰的な合成の例


(a)

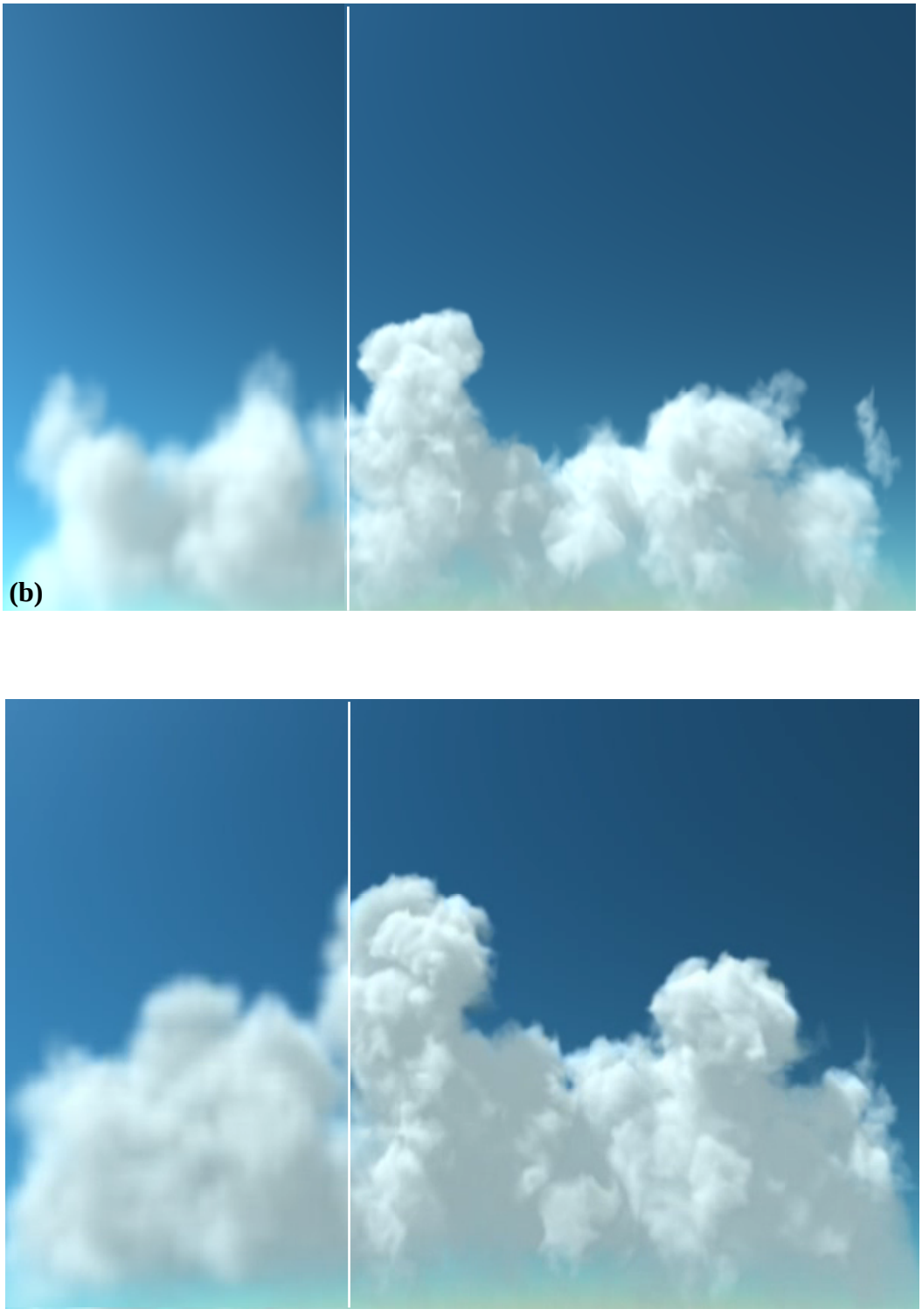


図 4.6: 煙 (a) および雲 (b) の例

表 4.1: 格子数

Example	n_{2D}	n_{low}	n_{high}
fire (GPU)	128×256	$32 \times 32 \times 64$	$128 \times 128 \times 256$
smoke (GPU)	256×128	$64 \times 32 \times 32$	$256 \times 128 \times 128$
clouds (CPU)	384×384	$96 \times 96 \times 60$	$384 \times 384 \times 240$

表 4.2: 計算時間

Example	T_{pre}	T_{low}	T_{synth}	T_{high}
fire (GPU)	48.72 sec	0.05 sec	0.34 sec	0.91 sec
smoke (GPU)	48.68 sec	0.04 sec	0.29 sec	0.85 sec
clouds (CPU)	399.89 sec	0.97 sec	72.51 sec	-

4.6　まとめと今後の課題

本手法では低解像度のシミュレーションから高解像度の気体現象のアニメーションを合成する手法を提案した．2 次元の速度場の前計算データベースを用い，データベースの構築には 2 次元シミュレーションを一度だけ実行すればよい．また，炎，煙，雲において合成が可能であることが確認できた．

今後の課題としては，高解像度の速度場やスカラ量に割くメモリ量の削減があげられる．本手法では，2 次元の速度場データを用いることで，ランタイム処理でのメモリ使用量の低減も実現した．しかし，高解像度における速度場やスカラ場を持つ必要があり，より高解像度な結果を作成する場合，メモリ量が膨大なものとなる．これに対して，粒子を用いてスカラ値のトレースを行うことで，高解像度の格子の利用を抑えることが考えられる．また，図 4.4(b) の障害物を考慮した結果において，物体内部に速度や温度が入り込んでしまう問題がある．これに関しては，合成された速度場および移流後の温度場，密度場に対して境界条件を適用するなどの対処が考えられる．

第5章　流れ場のバリエーション生成

映画やゲームなどの映像作品では，大規模なシーンを作るために，多数の流体映像を用意しなければならない場合がある．このような大規模なシーンの例としては，村で複数の家が火事になっているシーンやミサイル攻撃により複数の爆発が発生しているシーンなどが挙げられる．これらのシーンでは，動きは異なるが類似している複数の流体アニメーションが必要になる．しかし，このような類似した複数のアニメーションをシミュレーションのパラメータ調整で作成するのは非常に困難である．また，流体シミュレーションの計算コストの高さも問題となる．そこで，本章では，シミュレーションにより作成された単一のデータセットから，動きの異なる様々なアニメーションを生成するための手法を提案する．提案法により，大規模なシーンを低コストで作成することが可能である．

5.1　研究背景

映画やゲームなどのエンターテインメントアプリケーションでは，類似した流体アニメーションが必要とされる場合がある．例としては，ミサイル攻撃による複数の爆発，複数の川の流れ，火事になっている複数の家，複数の煙突から立ち上る煙などのシーンが挙げられる．このようなシーンを作る際，同一のアニメーションを繰り返し使用すると，合成されるシーンのリアリティを損なってしまう．そのため，映像を制作するアニメータは，動きの異なる複数のアニメーションを作成する必要がある．これは，異なるパラメータ設定で何度も流体シミュレーションを繰り返すことで達成できる．しかし，そのような類似したアニメーションを作成するために，シミュレーションパラメータを調整することは，非常に困難である．また，膨大な計算コストも必要

となる．この問題を解決することが，本研究の目標である．

手続的な手法は，比較的低い計算コストで類似した複数の流れを生成することができる．例えば，低コストで，ユーザ所望の炎アニメーションを作成するための手法がいくつか提案されている [13, 21]．これらの手法では，炎の通り道を表す曲線を変形することで炎のアニメーションをユーザが任意にデザインすることができる．また，乱流ノイズ関数を用いて，流れのモデリングを行うための手続的な手法もいくつか提案されている [4, 28]．これらの手法を用いることで，アニメータは簡単に乱流のような動きを作成することができる．また，類似した複数のアニメーションを低コストで作成することもできる．しかし，これら手続的な手法は流れ場の生成に流体シミュレーションを用いていないため，物理ベースのシミュレーションを用いた場合に比べ，写実性に欠けたアニメーションとなってしまう．

上記の問題を解決するため，本手法では単一のシミュレーションデータから様々なバリエーションの流体アニメーションを生成するための手法を提案する．提案法は，煙や炎などの非圧縮性流体のアニメーションの生成に適している．また，バリエーションは周波数および空間領域において生成され，ユーザはバリエーションの度合いを指定する．そして，結果の流れ場を用い，煙の密度などのスカラー量を移流させ，結果画像を生成する．

提案手法の主な特徴は，入力となる流体の速度場を，非圧縮性を持つ基底関数を用いて表現することである．非圧縮性を持つ基底関数として，提案法ではラプラシアン固有関数を用いる [40]．そして，流れ場をラプラシアン固有関数で展開した際の係数をランダムに変調することでバリエーションを生成する．さらに，本提案システムでは，シミュレーション空間のサイズを変更することで，空間領域でのバリエーションを生成する．このリサイズされたシミュレーション空間での流れ場は最適化を行うことで算出する．また，これら 2 つの手法を組み合わせることで，周波数および空間領域両方のバリエーションを生成することができる．

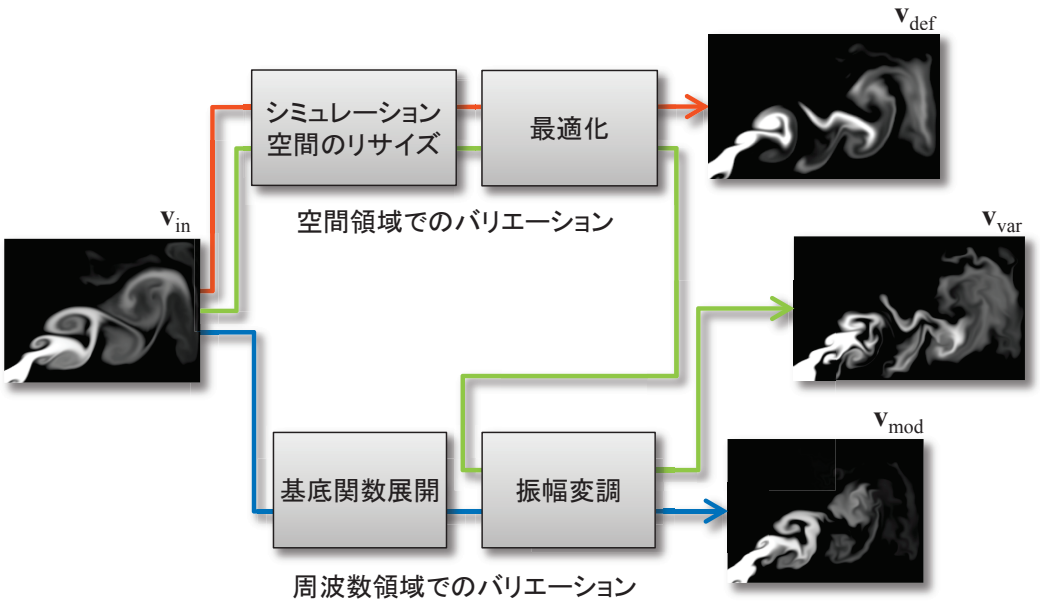


図 5.1: 提案手法の概要

5.2 提案手法

図 5.1 に提案法の概要を示す．本手法では，バリエーションを生成するために 2 種類の処理を行う．1 つ目は，振幅変調を行うことで，周波数領域での流れ場のバリエーション $\mathbf{v}_{\mathrm{mod}}(t_n)$ を生成する．この処理は，入力となる速度場 $\mathbf{v}_{\mathrm{in}}(t_n)$ を非圧縮な基底関数で展開した際の展開係数を変調する．もう一方は，シミュレーション空間をリサイズすることで，空間領域での流れ場のバリエーション $\mathbf{v}_{\mathrm{def}}(t_n)$ を生成する．この処理では，シミュレーション空間のサイズをランダムに変更する．本提案システムでは，リサイズされた速度場と結果の速度場との差分を目的関数とする最小化問題を解き，流れ場を生成する．また，これら 2 つの処理は組み合わせることができ，周波数および空間領域両方におけるバリエーション $\mathbf{v}_{\mathrm{var}}(t_n)$ を生成することができる．以下では，上記の処理についてその詳細を説明する．

5.2.1 周波数領域でのバリエーション

まず，入力となる流れ場 $\mathbf{v}_{\mathrm{in}}(t_n)$ を非圧縮な基底関数 $\mathbf{\Phi}_i$ の線形和で以下のように表現する．

$$\mathbf{v}_{\mathrm{in}}(t_n) = \sum_{i=0}^{N-1} w_i(t_n)\mathbf{\Phi}_i, \tag{5.1}$$

ここで，N は基底関数の数，$t_n, (n = 0, 1, \cdots)$ は離散化されたタイムステップを表す．本手法では，非圧縮性の基底関数 $\mathbf{\Phi}_i$ としてラプラシアン固有関数を用いる [40]．この基底関数は流れ場に対して，非圧縮性の条件を強制する．i 番目の基底関数に対する係数 $w_i(t_n)$ は，以下の式を解くことで計算される．

$$w_i(t_n) = \int_{\mathbf{x}\in\Omega} \mathbf{v}_{\mathrm{in}}(t_n) \cdot \mathbf{\Phi}_i d\mathbf{x}, \tag{5.2}$$

ここで，Ω は入力の流れ場における領域全体，$\cdot$ は 2 つのベクトルの内積を表わす．

この係数 $w_i(t_n)$ を変調することで，周波数領域における流れ場のバリエーション $\mathbf{v}_{\mathrm{mod}}(t_n)$ を生成する．提案法では，次式により流れ場の各周波数数成分に対する係数 $w_i(t_n)$ を変調する．

$$\mathbf{v}_{\mathrm{mod}}(t_n) = \sum_{i=0}^{N-1} g_i w_i(t_n)\mathbf{\Phi}_i, \tag{5.3}$$

ここで，g_i は $w_i(t_n)$ を変調する利得を表わし，乱数により値が決定される．

5.2.2 空間領域でのバリエーション

この節では，空間領域での流れ場のバリエーション生成法について説明する．提案法は，任意にシミュレーション空間を伸張もしくは収縮することで，シミュレーション空間のサイズを変更する．図 5.2 にシミュレーション空間をリサイズする方法を示す．まず，入力の速度場を切断する方向とラインを乱数により決定する．図 5.2 では，切断方向を水平方向に設定した場合である．次に，シミュレーション空間を伸張，収縮させる長さを乱数により決定する．伸張処理では，シミュレーション空間を切断線により 2 つの領域に分割する．そして，これら 2 つの領域を分離し，2 つの領域の間に新たな格子点を挿入する (図 5.2)．収縮処理の場合，2 つの切断線を指定し，そして 2 つの

切断線間の格子点を図 5.2 に示すように除去する．シミュレーション空間のリサイズ後，提案法はリサイズされたシミュレーション空間と同じサイズの新しい格子空間を用意する．新しい格子空間における格子点の数は N_{def} とする．そして，入力の速度をシミュレーション空間のリサイジングの情報に従って，新しい格子空間にコピーする．以降リサイズされた速度場を $\mathbf{v'}_{\mathrm{in}}$ と表記する．上記の処理後，リサイズされた空間において定義された基底関数に対する重み係数 $\mathbf{w'}(t_n) = (w'_0(t_n), w'_1(t_n), \cdots, w'_{N-1}(t_n))$ を以下の最小化問題を解くことで計算する．

$$\arg\min_{\mathbf{w'}(t_n)}(E(t_n) + \alpha \sum_{i=0}^{N-1} w'^2_i(t_n)), \tag{5.4}$$
$$E(t_n) = \sum_{\mathbf{x}\in\Omega_{in}} |\mathbf{v'}_{\mathrm{in}}(\mathbf{x}, t_n) - \sum_{i=0}^{N-1} w'_i(t_n)\mathbf{\Phi'}_i(\mathbf{x})|^2,$$

ここで，Ω_{in} は $\mathbf{v'}_{\mathrm{in}}$ 上に入力の速度 $\mathbf{v}_{\mathrm{in}}$ が割り付けられている領域である．また，α は第 2 項の正則化項の効果を調整するために用いられるユーザ指定の係数である．$\mathbf{\Phi'}_i$ はリサイズ後のシミュレーション空間上に定義され，格子によりサンプリングされる．

$E(t_n)$ は，リサイズされた入力の速度場と結果の流れ場との間の差分を意味する．上記の方程式を解くことで，空間領域での流れ場のバリエーション $\mathbf{v}_{\mathrm{def}}(t_n)$ を合成する．

$$\mathbf{v}_{\mathrm{def}}(t_n) = \sum_{i=0}^{N-1} w'_i(t_n)\mathbf{\Phi'}_i. \tag{5.5}$$

前の段落において定義した誤差関数 $E(t_n)$ により，式 (5.4) の最小化問題を解くことができる．各係数 $w'_i(t_n)$ において式 (5.4) の微分をとることで，以下の行列方程式を得る．

$$(\mathbf{A} + \alpha\mathbf{I})\mathbf{w'}(t_n) = \mathbf{c}(t_n), \tag{5.6}$$

ここで，$\mathbf{A}$ および $\mathbf{I}$ は $N \times N$ の行列であり，$\mathbf{w'}(t_n)$ および $\mathbf{c}(t_n)$ は N 次元の列ベクトルである．また，$\mathbf{I}$ は単位行列である．$\mathbf{A}$ および $\mathbf{c}$ は誤差関数 E に関係している．行列 $\mathbf{A}$ の (i, j) 番目の要素 a_{ij} および列ベクトル $\mathbf{c}(t_n)$ の i 番目の要素 c_i は，それぞれ以

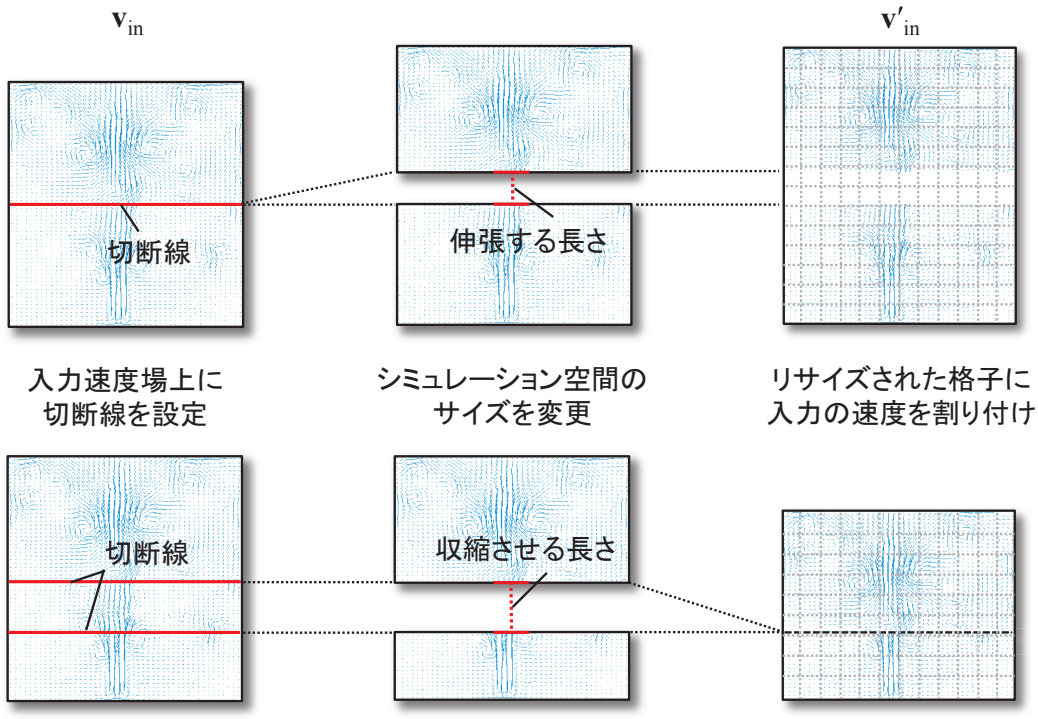


図 5.2: 入力速度場の伸張（上段）および収縮（下段）

下の式で与えられる．

$$
\begin{aligned}
a_{ij} &= \sum_{\mathbf{x}\in\Omega_{\text{in}}} \mathbf{\Phi'}_i(\mathbf{x}) \cdot \mathbf{\Phi'}_j(\mathbf{x}), \\
c_i &= \sum_{\mathbf{x}\in\Omega_{\text{in}}} \mathbf{v'}_{\text{in}}(\mathbf{x}, t_n) \cdot \mathbf{\Phi'}_i(\mathbf{x}),
\end{aligned}
$$

上記方程式における関数間の内積を計算するために，$\mathbf{v'}_{\text{in}}$ および $\mathbf{\Phi'}_i$ はリサイズされたシミュレーション空間の格子上でサンプルされる．そして，そのサンプルされた値を用いることで，内積を近似的に計算する．

重み係数ベクトル $\mathbf{w}'(t_n)$ は式 (5.6) の行列方程式を解くことで得られる．この方程式を効率的に解くために，提案法では LU 分解法 [30] を用いる．行列 $(\mathbf{A}+\alpha\mathbf{I})$ の LU 分解は高速に計算でき，分解された L 行列，U 行列を用いることで重みベクトル $\mathbf{w}'(t_n)$ を効率的に算出できる．

5.3 実験結果

本節では，提案法を用いて作成したいくつかの実験例を示す．実験環境は，CPU が Intel Core i7 2600K（メモリ 16GB），GPU が NVIDIA GeForce GTX 680 となっている．パラメータ設定と各結果の計算時間を表 5.1 にまとめる．N は基底関数の数，N_{def} はリサイズ後の速度場の格子数，T は流れ場を更新するために係る時間であり，単位は ms/フレーム である．図 5.4 および図 5.5，5.6 の例では，256×256 の格子点上で計算された入力速度場を用いた．また図 5.3 では，入力の速度場は 64×64 の格子上で計算される．

図 5.3 は，他のリサイジング手法により生成された結果と提案法により生成された結果との比較を示す．図 5.3(a) は 64×64 の格子上で計算された入力の流れ場である．切断線は，図 5.3(a) に赤色の点線として示す．提案法により生成された結果は，図 5.3(b) に示す．結果からもわかるとおり，本手法では，連続な流れ場かつ入力の流れ場に類似した流れを生成することに成功した．図 5.3(c) は，入力の流れ場をリサイズされたシミュレーション空間にワーピングした流れ場である．この場合，流れ場がオリジナルの流れから変形してしまっている．図 5.3(d) はリサイズされたシミュレーション空間に置いて流れ場を再度シミュレーションした場合の例である．ただし，シミュレーションパラメータは図 5.3(a) の設定と同様である．この場合，煙がシミュレーション空間の上部の領域まで上昇していない．そのため，試行錯誤的な作業を通して，シミュレーションパラメータを適切に調整しなければならない．

次に，図 5.4 は，振幅変調の例を示す．流れ場は，煙の密度を移流させることで，可視化を行っている．煙の発生源は，シミュレーション空間の左下角に配置している．図 5.4(a) は，入力となる煙のアニメーションを示す．図 5.4(b) から (d) は，入力のアニメーションの振幅を変調することで生成されたバリエーションの結果である．(b) の結果では，入力の速度場の高周波成分を増加させるよう変調を行っている．また (c) では，低周波成分を減少させている．(d) の例では，(b) および (c) の両方の変調を適用して生成した結果である．また，左列から右列にかけて一定時間時間ずつ経過した際

の結果を示した．実験結果から提案法により周波数領域でのバリエーションが生成できているのがわかる．

最後に，シミュレーション空間をリサイズし，空間領域でのバリエーション生成を行った結果を図 5.5，5.6 に示す．まず，入力のアニメーションを伸張させた場合の結果について述べる．図 5.5(a) は，シミュレーションにより作成した入力の流れ場である．切断線は 5.5(a) 上に赤色の点線で示す．図 5.5(b) および (c) は，(a) のシミュレーション空間を伸張した結果である．伸張させる量は，入力のシミュレーション空間における鉛直方向のた高さの 4 分の一に設定した．さらに，図 5.5(c) の結果では，(b) に対し振幅変調も加えて適用した結果である．この際，流れ場の高周波成分を強調するよう変調を行っている．次に，シミュレーション空間を収縮させた場合の結果を示す．図 5.6(a) は，入力とした煙のアニメーションである．2 つの切断線を 5.6(a) 上に赤色の点線で示す．図 5.6(b) および (c) は，(a) のシミュレーション空間を収縮させた結果である．収縮させる量は，伸張の際と同様，入力のシミュレーション空間における鉛直方向のた高さの 4 分の一に設定した．さらに，図 5.6(c) の結果では，(b) に対しさらに振幅変調を適用した．こちらも，流れ場の高周波成分を強調するよう変調を行っている．また，図 5.5，5.6 のどの結果においても，一定時間ずつ経過した結果画像を中段，下段に示した．伸張，収縮どちらの例についても，提案法を用いることで，連続した流れ場を生成することができているのがわかる．また，周波数および空間両方の領域におけるバリエーションが生成できていることもわかる．

表 5.1: パラメータ設定と計算時間

	N	N_{def}	T
Fig. 5.4	1024	–	120
Fig. 5.3(b)	400	64×96	10
Fig. 5.5(b)(c)	1024	256×320	140
Fig. 5.6(b)(c)	1024	256×192	80

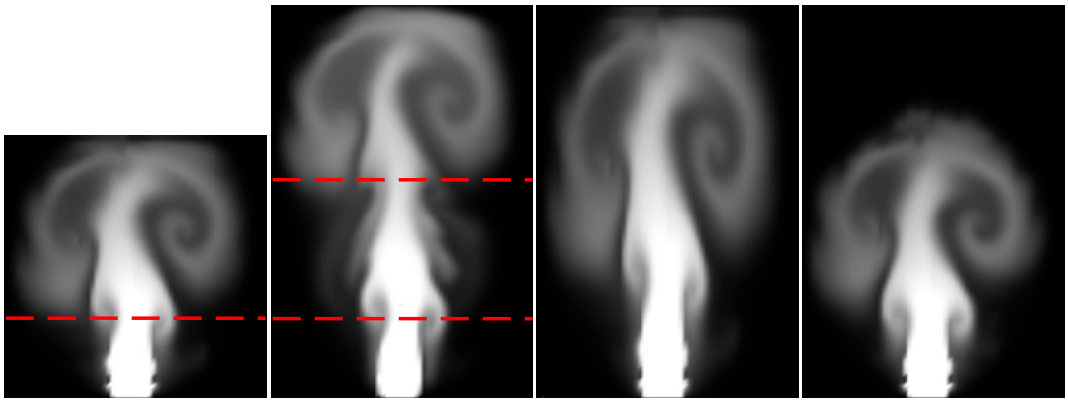


図 5.3: 他のリサイジング手法を用いた結果との比較

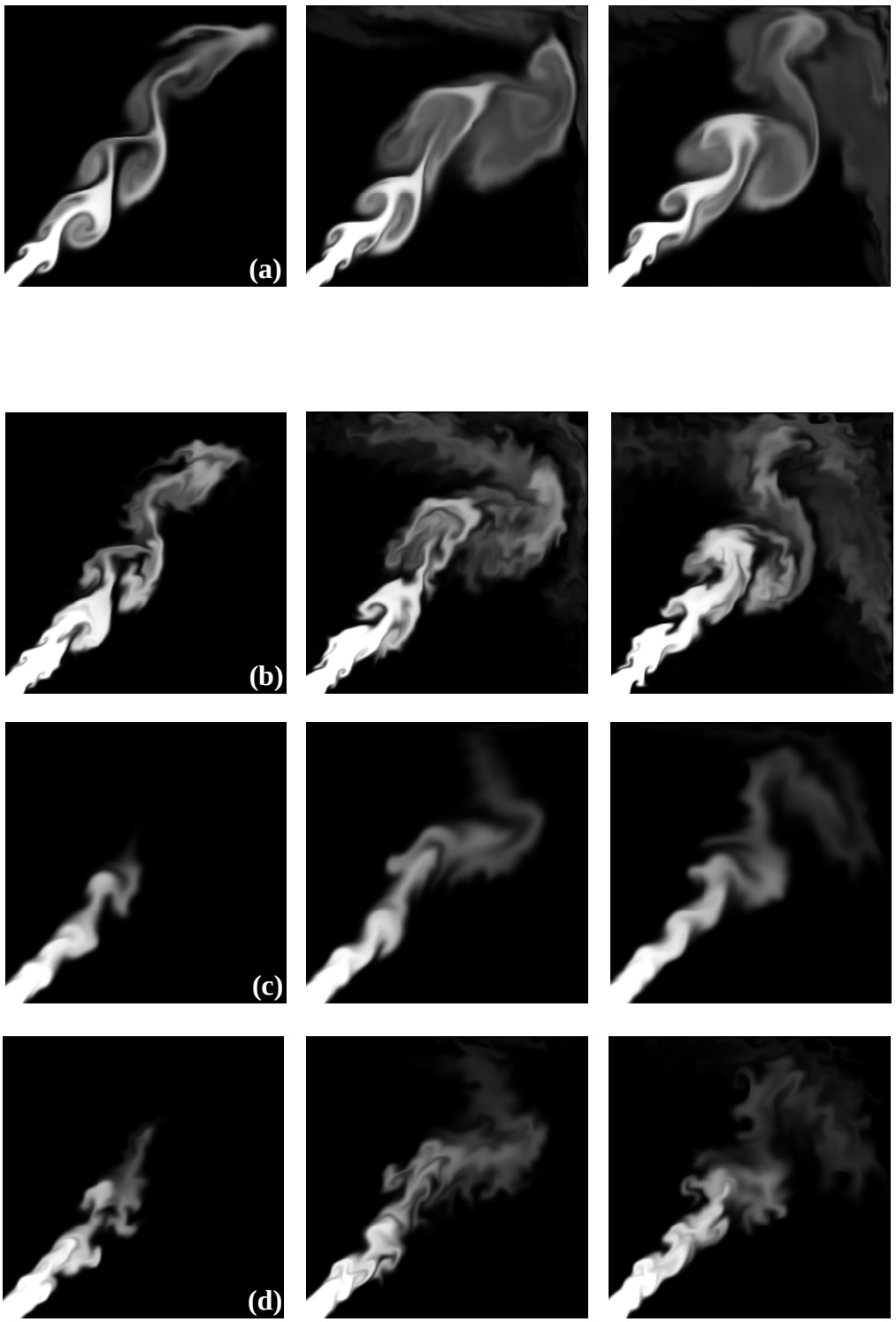


図 5.4: 周波数領域でのバリエーション生成結果

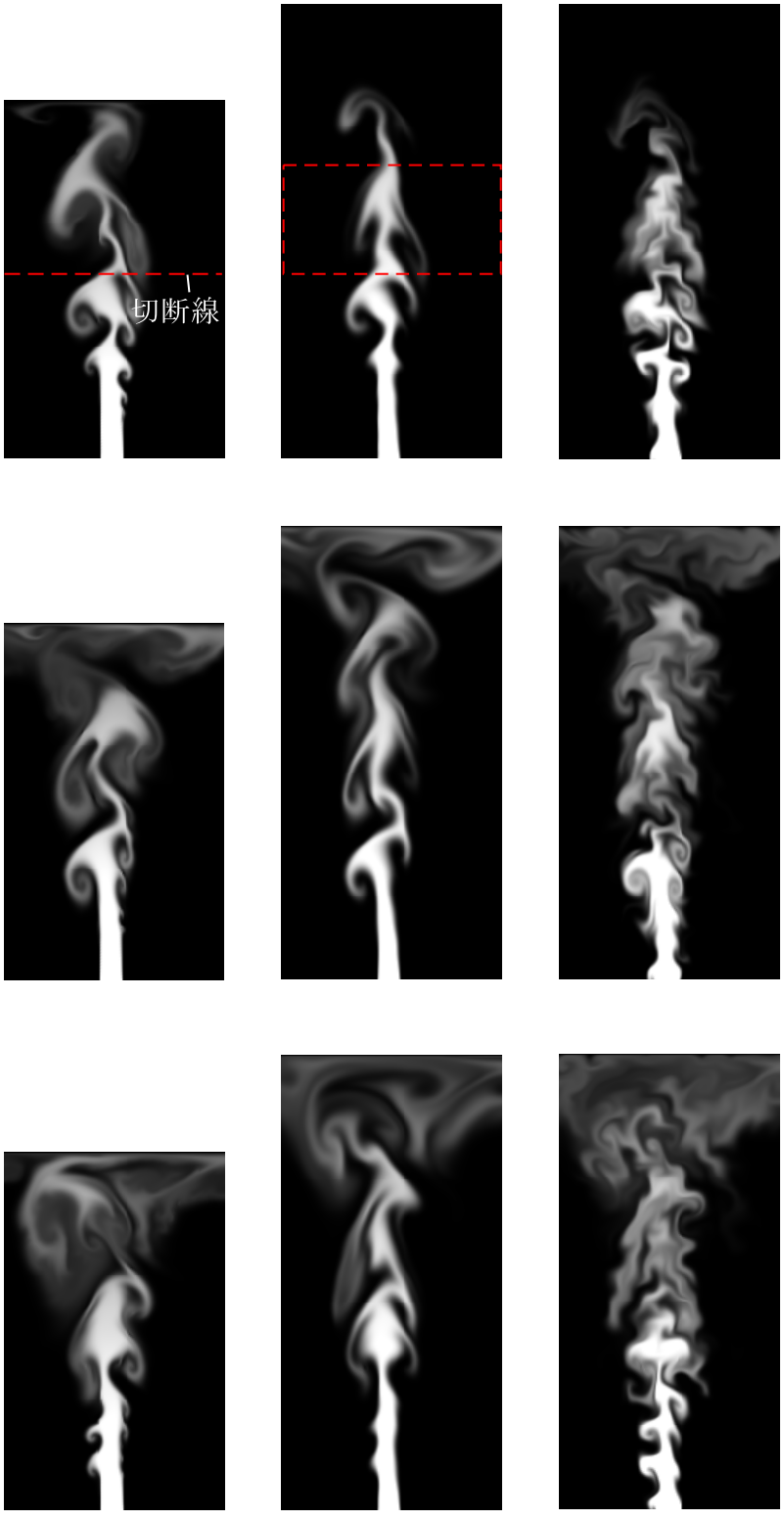


図 5.5: 空間領域でのバリエーション生成結果（伸張）

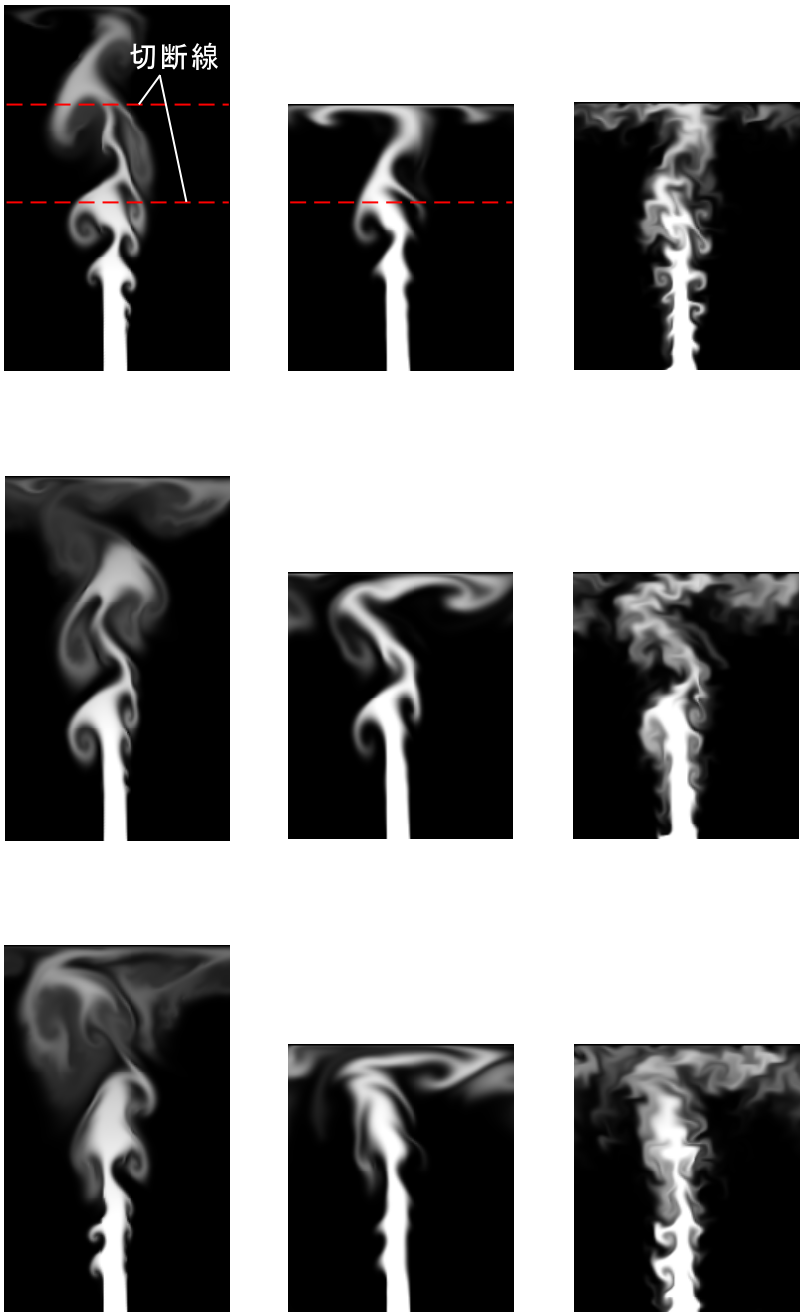


図 5.6: 空間領域でのバリエーション生成結果（収縮）

5.4 まとめと今後の課題

本章では，流体流れ場のバリエーションを合成するための方法を提案した．提案法では，周波数および空間領域において，バリエーションを生成することができる．流れ場をラプラシアン固有関数により表現し，周波数領域における流れ場のバリエーションは，基底関数に対する重み係数を変調することで生成した．また，空間領域での流れ場のバリエーションは，伸張，収縮させた後の入力の速度場を目的関数とする最小化問題を解くことで生成することができる．本章では，いくつかの例を示すことで，提案法がバリーションを生成することに対し有用であることを示した．今後，本提案手法を 3 次元の流れ場に拡張することを予定している．

提案法の問題点の一つとして，計算コストが基底関数の数に比例するという点がある．本手法により生成される流れ場における詳細の度合いは，用いる基底関数の数に比例する．もし，使用する基底関数の数が多いと，$\mathbf{A}$ と $\mathbf{c}(t_n)$ の算出および式 (5.3) と (5.5) を用いた流れ場の再構築の計算に膨大な計算コストが必要となる．そのため，ユーザは計算およびストレージのコストの増加を犠牲にすれば，詳細な流れ場のバリエーションを作成することができる．

その他の問題点として，g_i による変調の度合いが大きすぎる場合，本手法により生成される流れ場は流体の物理法則に従っていない場合がある．この問題を解決するために，提案法により生成された速度場に対し Navier-Stokes 方程式を計算し，その後，提案法により生成された流れ場と Navier-Stokes 方程式を計算した場合の速度場を比較する．この比較を行うことにより，提案法により生成された流れ場が流体の物理法則に従っているかどうかを評価することができる．今後，これらの流れ場を評価する実験を行う予定である．

第6章　結論

本論文では，映画やゲームなどで流体映像を制作する際に，流体シミュレーションに係る計算コストやパラメータ調整等にかかる作業量を削減し，効率的に所望の流体映像を生成するための手法を提案した．

近年の計算機の飛躍的な性能向上により，一般の PC でも写実的な CG 映像を作成することが可能となり，また，GPU による並列計算の普及および GPU の性能向上により，写実的な流体映像を作成することも出来るようになってきた．これにより，映画やゲームにおいて，現実と見紛うほどの映像を作成することも可能となってきた．しかし，このような写実的な映像を作成するためには，いまだに高い計算コストが必要である．また，映画やゲームの各シーンに合うような，所望の流体映像を作成するためには，シミュレーションパラメータを試行錯誤的に調整しなければならない．そのため，一つ一つのシーンの作成に膨大な時間がかかってしまっているのが現状である．また，戦争シーンや火事のシーンなどの大規模なシーンでは，複数の動きの異なる流体映像が必要となる．しかし，パラメータを少しずつ変更し，動きの異なる流体映像を多数作成するには，極めて膨大な時間を要してしまう．

本論文では，これらの問題を解決するために，3 つの手法を提案した．まず，爆発強度の最適化と予測制御の 2 つの処理を用いることで，これまで特定の形状として表現することが困難であった爆発のシミュレーションを制御する手法を提案した．また，流体シミュレーション結果をデータベースを作成し，データベース内のデータの線形和で流れ場を表現することで，低品質な流れ場から高品質な流れ場へ変換する手法を提案し，パラメータ調整に係る時間の低減を実現した．さらに，大規模なシーンの作成に対し，単一のシミュレーションデータから，シミュレーションの実行をせずに様々

なバリエーションを持った流体映像を作成する手法を提案した．この手法では，流れ場を非圧縮な基底関数の線形和で表現することで，バリエーション生成を行った．

各章において，今後の課題については述べたが，ここでもう一度各手法の今後の課題について簡単にまとめる．

(1) 爆発シミュレーションの制御（第 3 章）: 本手法では，2 次元のシミュレーションに対して制御を行い，3 次元のシミュレーションに関しては，2 次元の組み合わせとして擬似的に表現した．しかし，2 次元の組み合わせでは，3 次元的な乱流成分が表現できない場合がある．そのため，提案手法を 3 次元のシミュレーションに拡張する必要がある．

(2) 流体シミュレーションの超解像処理（第 4 章）: 2 次元のシミュレーションデータベースを利用することで，低解像度から高解像度の結果へ低コストで変換することができた．しかし，関連研究でも言えることだが，高解像度の格子を用意する必要があり，より写実的な結果を生成したい場合メモリ容量が膨大になる．これに対して，粒子を用いることで高解像度の格子の利用を抑えることが今後の課題である．

(3) 流体シミュレーションのバリエーション生成（第 5 章）: 非圧縮な基底関数により流れ場を表現することで，様々なバリエーションの流体映像を作成可能とした．しかし，詳細な流れ場について本手法を適用する場合，多くの基底関数が必要となり，計算およびストレージのコストが膨大になる．そのため，ウェーブレット基底等のローカルサポートな基底関数を用いることでこの問題を解決可能か実験する必要がある．

謝辞

本研究および本論文の作成に関し，多大なる御指導，御討論を頂きました北海道大学大学院情報科学研究科メディアネットワーク専攻土橋宜典准教授に心より感謝を申し上げます.

また，貴重な御意見，御討論を頂きました北海道大学大学院情報科学研究科メディアネットワーク専攻山本強教授，青木直史助教に心より感謝を申し上げます．あわせて，本研究に対し，貴重な御意見，御討論を頂きました株式会社オー・エル・エム・デジタル研究開発部門安生健一氏，九州大学マス・フォア・インダストリ研究所落合啓之教授，和歌山大学システム工学部岩崎慶准教授に深く感謝いたします.

本論文をまとめるにあたり，御助言，御指導を頂いた北海道大学大学院情報科学研究科メディアネットワーク専攻，長谷山美紀教授に心より感謝いたします.

本論文をまとめるにあたり，御助言，御指導を頂いた北海道大学大学院情報科学研究科メディアネットワーク専攻，荒木健治教授に心より感謝いたします.

本研究に対して，御協力を頂いた北海道大学大学院情報科学研究科メディアネットワーク専攻情報メディア学講座情報メディア環境学研究室諸氏に心より御礼を申し上げます.

参考文献

[1] S. Baker and T. Kanade. Hallucinating faces. In *IEEE International Conference on Automatic Face and Gesture Recognition*, pp. 83–88, 2000.

[2] C. Batty, F. Bertails, and R. Bridson. A fast variational framework for accurate solid-fluid coupling. *ACM Transactions on Graphics*, Vol. 26, No. 3, p. Article 100, 2007.

[3] R. Bridson. *Fluid Simulation for Computer Graphics*. AK Peters, 2008.

[4] Robert Bridson, Jim Hourihan, and Marcus Nordenstam. Curl-noise for procedural fluid flow. *ACM Transactions on Graphics*, Vol. 26, No. 3, p. Article 46, 2007.

[5] K. Crane, I. Llamas, and S. Tariq. *Real Time Simulation and Rendering of 3D Fluids*, chapter 30. Addison-Wesley, 2007.

[6] Y. Dobashi, K. Kusumoto, T. Nishita, and T. Yamamoto. Feedback control of cumuliform cloud formation based on computational fluid dynamics. *ACM Transactions on Graphics*, Vol. 27, No. 3, p. Article 94, 2008.

[7] Y. Dobashi, Y. Matsuda, T. Yamamoto, and T. Nishita. A fast simulation method using overlapping grids for interactions between smoke and rigid objects. *Computer Graphics Forum*, Vol. 23, No. 3, pp. 539–546, 2008.

[8] R. Fattal and D. Lischinski. Target-driven smoke animation. *ACM Transactions on Graphics*, Vol. 23, No. 3, pp. 439–446, 2004.

[9] R. Fedkiw, J. Stam, and H. W. Jansen. Visual simulation of smoke. In *Proceedings of ACM SIGGRAPH 2001*, pp. 15–22, 2001.

[10] B. E. Feldman, J. F. O'Brien, and O. Arikan. Animating suspended particle explosions. In *Proceedings of ACM SIGGRAPH 2003*, pp. 708–715, 2003.

[11] B. E. Feldman, J. F. O'Brien, and B. M. Klingner. Animating gases with hybrid meshes. *ACM Transactions on Graphics*, Vol. 24, No. 3, pp. 904–909, 2005.

[12] N. Foster and R. Fedkiw. Practical animation of liquids. In *Proceedings of ACM SIGGRAPH 2001*, pp. 23–30, 2001.

[13] A. R. Fuller, H. Krishnan, K. Mahrous, B. Hamann, and K. I. Joy. Real-time procedural volumetric fire. In *Proceeding of the 2007 symposium on Interactive 3D graphics and games*, pp. 175–180, 2007.

[14] J. K. Hodgins G. D. Yngve, J. F. O'Brien. Animating explosions. In *Proceedings of ACM SIGGRAPH 2000*, pp. 29–36, 2000.

[15] J. Hong and C. Kim. Controlling fluid animation with geometric potential. *Computer Animation and Virtual Worlds*, Vol. 15, No. 3-4, pp. 147–157, 2004.

[16] C. Horvath and W. Geiger. Directable, high-resolution simulation of fire on the gpu. *ACM Transactions on Graphics*, Vol. 28, No. 3, p. Article 41, 2009.

[17] B. Kang, Y. Jang, and I. Ihm. Animation of chemically reactive fluids using a hybrid simulation method. In *SCA '07: Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 199–208, 2007.

[18] B. Kim, Y. Liu, I. Llamas, X. Jiao, and J. Rossignac. Simulation of bubbles in foam with the volume control method. *ACM Transactions on Graphics*, Vol. 26, No. 3, p. Article 98, 2007.

[19] Theodore Kim, Nils Thurey, Doug James, and Markus Gross. Wavelet turbulence for fluid simulation. *ACM Transactions on Graphics*, Vol. 27, No. 3, p. Article 3, 2008.

[20] Y. Kim, R. Machiraju, and D. Thompson. Path-based control of smoke simulations. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 33–42, 2006.

[21] A. Lamorlette and N. Foster. Structural modeling of flames for a production environment. *ACM Transactions on Graphics*, Vol. 21, No. 3, pp. 729–735, 2002.

[22] M. Lentine, W. Zheng, and R. Fedkiw. A novel algorithm for incompressible flow using only a coarse grid projection. *ACM Transactions on Graphics*, Vol. 29, No. 4, p. Article 114, 2010.

[23] F. Losasso, F. Gibou, and R. Fedkiw. Simulating water and smoke with an octree data structure. *ACM Transactions on Graphics*, Vol. 23, No. 3, pp. 457–462, 2004.

[24] A. McNamara, A. Treuille, Z. Popovic, and J. Stam. Fluid control using the adjoint method. *ACM Transactions on Graphics*, Vol. 23, No. 3, pp. 449–456, 2004.

[25] D. Q. Nguyen, R. Fedkiw, and H. W. Jensen. Physically based modeling and animation of fire. *ACM Transactions on Graphics*, Vol. 21, No. 3, pp. 721–728, 2002.

[26] Michael B. Nielsen and Brian B. Christensen. Improved variational guiding of smoke animations. *Computer Graphics Forum*, Vol. 29, No. 2, pp. 705–712, 2010.

[27] Michael B. Nielsen, Brian B. Christensen, Nafees Bin Zafar, Doug Roble, and Ken Museth. Guiding of smoke animations through variational coupling of simulations at different resolutions. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 217–226, 2009.

[28] Mayur Patel and Noah Taylor. Simple divergence-free fields for artistic simulation. *Journal of Graphics, GPU, and Game Tools*, Vol. 10, No. 4, pp. 49–60, 2005.

[29] Tobias Pfaff, Nils Thuerey, Jonathan Cohen, Sarah Tariq, and Markus Gross. Scalable fluid simulation using anisotropic turbulence particles. *ACM Transactions on Graphics*, Vol. 29, No. 6, p. Article 174, 2010.

[30] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing.* Cambridge University Press, 2007.

[31] Y. Dobashi R. Miyazaki and T. Nishita. Simulation of cucumuliform clouds based on computational fluid dynamics. In *EUROGRAPHICS 2002 Short Presentations*, pp. 405–410, 2002.

[32] N. Rasmussen, D. Q. Nguyen, W. Geiger, and R. Fedkiw. Smoke simulation for large scale phenomena. *ACM Transactions on Graphics*, Vol. 22, No. 3, pp. 703–707, 2003.

[33] H. Schechter and R. Bridson. Evolving sub-grid turbulence for smoke animation. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pp. 1–7, 2008.

[34] L. Shi and Y. Yu. Taming liquids for rapidly changing targets. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 229–236, 2005.

[35] Jos Stam. Stable fluids. In *Proceedings of ACM SIGGRAPH 1999, Annual Conference Series*, pp. 121–128, 1999.

[36] N. Thürey, R. Keiser, M. Pauly, and U. Rüde. Detail-preserving fluid control. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 7–12, 2006.

[37] A. Treuille, A. McNamara, Z. Popovic, and J. Stam. Keyframe control of smoke simulations. *ACM Transactions on Graphics*, Vol. 22, No. 3, pp. 716–723, 2003.

[38] Adrien Treuille, Andrew Lewis, and Zoran Popovic. Model reduction for real-time fluids. *ACM Transactions on Graphics*, Vol. 25, No. 3, pp. 826–834, 2006.

[39] Martin Wicke, Matt Stanton, and Adrien Treuille. Modular bases for fluid dynamics. *ACM Transaction on Graphics*, Vol. 28, No. 3, p. Article 39, 2009.

[40] Tyler De Witt, Christian Lessig, and Eugene Fiume. Fluid simulation using laplacian eigenfunctions. *ACM Transactions on Graphics*, Vol. 31, No. 1, p. Article 10, 2012.

[41] Zhi Yuan, Fan Chen, and Ye Zhao. Pattern-guided smoke animation with lagrangian coherent structure. *ACM Transactions on Graphics*, Vol. 30, No. 6, p. Article 136, 2011.

研究業績 学会誌

[1] 佐藤周平, 土橋宜典, 山本強, 安生健一, “最適な初期強度分布の推定および予測制御による爆発シミュレーションの制御”, 映像情報メディア学会誌, Vol.65, No.10, pp.98-103, (Oct. 2011).

[2] 佐藤周平, 森田拓也, 土橋宜典, 山本強, “基本速度場による流体アニメーションの高解像度化”, 電子情報通信学会 論文誌 D, Vol.J96-D, No.2, pp.338-345, (Feb. 2013).

査読付国際会議

[1] Yoshinori Dobashi, Shuhei Sato, Tsuyoshi Yamamoto, Ken Anjyo, “Controlling Explosion Simulation”, SIGGRAPH ASIA 2009 sketches, (Dec. 2009).

[2] Syuhei Sato, Yoshinori Dobashi, Tsuyoshi Yamamoto, Ken Anjyo, “Controlling Simulated Explosions by Optimization and Prediction”, CADGRAPHICS ’11 Proceedings of the 2011 12th International Conference on Computer-Aided Design and Computer Graphics, pp 296-301, (Sep. 2011).

[3] Syuhei Sato, Takuya Morita, Yoshinori Dobashi, Tsuyoshi Yamamoto, “A Data-Driven Approach for Synthesizing High-Resolution Animation of Fire”, DigiPro’12 Proceedings of the Digital Production Symposium, pp 37-42, (Aug. 2012).

[4] Syuhei Sato, Yoshinori Dobashi, Kei Iwasaki, Hiroyuki Ochiai, Tsuyoshi Yamamoto, “Generating Flow Fields Variations by Modulating Amplitude and Resizing Simulation Space”, SIGGRAPH ASIA 2013 Technical Briefs, (Nov. 2013).

国際会議

[1] Syuhei Sato, “Generating Variations of Flow Fields by Modulating Amplitude and Resizing Simulation Space”, Symposium MEIS2013, (Oct. 2013).

学会技術研究会

[1] 佐藤周平, 土橋宜典, 山本強, 安生健一, “流体力学に基づく爆発の動きのコントロール”，Visual Computing/グラフィクスと CAD 合同シンポジウム 2009，CD-ROM, (Jun. 2009).

[2] 佐藤周平，土橋宜典，山本強, 安生健一, “予測制御による爆発のキーフレームアニメーション”，情報処理学会グラフィックスと CAD 第 137 回研究会，研究報告, (Nov. 2009).

[3] 佐藤周平, 土橋宜典, 山本強, 安生健一, “初期強度分布の最適化による爆発のコントロール”，Visual Computing/グラフィクスと CAD 合同シンポジウム 2010, CD-ROM，(Jun. 2010).

[4] 佐藤周平, 土橋宜典, 山本強, “データベースを用いた炎のアニメーションの生成手法”，Visual Computing/グラフィクスと CAD 合同シンポジウム 2011，CD-ROM, (Jun. 2011).

[5] 佐藤周平，土橋宜典，山本強, “基本速度場の合成による炎のアニメーション”，情報処理学会グラフィックスと CAD 第 145 回研究会，研究報告，(Nov. 2011).

[6] 森田拓也, 佐藤周平, 土橋宜典, 山本強, “基本速度場を用いた高解像度な流体映像の生成”，Visual Computing/グラフィクスと CAD 合同シンポジウム 2012，CD-ROM，(Jun. 2012).

[7] 佐藤周平，土橋宜典，山本強, “基本速度場の合成による炎のアニメーション”，情報処理学会グラフィックスと CAD 第 149 回研究会，研究報告，(Dec. 2012).

[8] 佐藤周平, 土橋宜典, 岩崎慶, 山本強, 落合啓之, “流体の流れ場のインタラクティブなデザイン”，Visual Computing/グラフィクスと CAD 合同シンポジウム 2013, CD-ROM，(Jun. 2013)．(グラフィクスと CAD 研究会優秀研究発表賞)

[9] 谷翼，土橋宜典, 佐藤周平，山本強, “ベクトル量子化を施した 3 次元テクスチャの合成”，情報処理学会グラフィックスと CAD 第 151 回研究会，研究報告，(Jun. 2013).

全国大会等発表

[1] 佐藤周平，土橋宜典，山本強, “流体力学に基づく爆発の動きのコントロール”，電子情報通信学会 2009 年総合大会講演論文集，D–11–14，(Mar. 2009).

[2] 佐藤周平，土橋宜典，山本強, “乱流成分を保持した爆発の動きのコントロール”，平成 21 年度電気・情報関係学会北海道支部連合大会講演論文集，CD-ROM，(Oct. 2009)．(若手優秀論文発表賞)

[3] 佐藤周平，土橋宜典，山本強, “データの合成による炎アニメーション生成”，平成 23 年度電気・情報関係学会北海道支部連合大会講演論文集，CD-ROM，(Oct. 2011).

[4] 森田拓也, 佐藤周平，土橋宜典，山本強, “スライスの合成に基づく雲の高解像度シミュレーション”，平成 23 年度電気・情報関係学会北海道支部連合大会講演論文集，CD-ROM，(Oct. 2011).

[5] 渋川雄平, 佐藤周平，土橋宜典，山本強, “流体シミュレーションを用いた炎のセル画調レンダリング”，平成 23 年度電気・情報関係学会北海道支部連合大会講演論文集，CD-ROM，(Oct. 2011).

[6] 水谷圭佑, 佐藤周平，土橋宜典，山本強, “流体シミュレーションに基づく炎の形状制御”，平成 25 年度電気・情報関係学会北海道支部連合大会講演論文集，CD-ROM，(Oct. 2013).